OVHcloud
Documentation

On-Prem Cloud Platform

Generated on 11 August 2026

Contents

Getting started

Network integration and connectivity for OPCP

Understand how OPCP integrates into your network — data plane, out-of-band (OOB) administration network, and which flows to allow per service model

Objective

On-Prem Cloud Platform (OPCP) is a private cloud solution deployed on your own infrastructure, delivered as an integrated stack (servers, switches, automation, control plane). Before and during deployment, your team needs to understand how OPCP interconnects with your enterprise network: which flows it sends, which services it consumes from you, and – if you have subscribed to the **Fully managed by OVHcloud** offering – how OVHcloud reaches the platform to operate it remotely.

The aim of this guide is to present OPCP's network architecture, distinguishing:

- the **out-of-band (OOB) administration network** used to operate the platform,
- the **data plane** used by the workloads hosted on the OPCP servers.

It is intended for both **architects** preparing a deployment and **administrators** taking over a platform that has already been delivered.

Requirements

- A general understanding of your enterprise network architecture (VLANs, addressing plan, edge devices, firewalls).
- Knowing the **OPCP subscription model** you have chosen (Self-managed or Fully managed by OVHcloud).
- Having reviewed the OPCP technical prerequisites you need to provide before deployment.

Instructions

1. OPCP service models

OPCP is available with two levels of delegation. The chosen model determines **who operates** the platform's control plane and therefore **which interconnection flows** must be opened toward OVHcloud.

Model	Who operates the Core (control plane)?	Who operates the application services?	Interconnection to OVHcloud required?
Self-managed	The customer	The customer	No
Fully managed by OVHcloud	OVHcloud	OVHcloud	**Yes** (IPsec)

Whatever the model, **managing what you deploy on top of the services provided by OPCP remains your responsibility**: virtual machines, containers, bare-metal instances, applications, business data, application-

level backups, and so on. OPCP provides the platform; you remain in charge of the use you make of it.

Whatever the model, the hardware architecture and the logical separation between networks are **identical**. The differences between Self-managed and Fully managed by OVHcloud relate to two points:

1. **The level of access to administration interfaces.** In Self-managed mode, your teams have full control of OPCP's management interfaces: OpenStack API, SSH access to the controllers, and two command-line tools embedded on the platform:
    - `opcp-cli`, which lets you **configure OPCP** (declaring platform resources, managing components, operational tasks),
    - `opcp-diag`, which produces a **platform health status** and **extracts the logs** required for remote diagnostics when an incident occurs.

    In Fully managed by OVHcloud mode, these interfaces are operated by the OVHcloud teams; your teams retain consultation and application-level operation access, but platform administration is delegated.
2. **The need for an IPsec link between your administration network and OVHcloud.** Required in Fully managed by OVHcloud mode so that OVHcloud teams can operate the platform remotely; this link does not exist in Self-managed mode.

2. The two OPCP network planes

OPCP relies on a strict separation between two networks with very different roles, equipment, and uses. **This separation is structural**: it underpins the security, resiliency, and quality of service of the platform.

The data plane

The data plane is the network **used by the workloads** hosted on the OPCP servers (virtual machines, containers, bare-metal instances exposed via OpenStack). It carries:

- the application traffic in and out of the instances,
- the communications between instances inside OPCP,
- the flows between OPCP and the rest of your production information system (databases, internal services, end users, and so on).

In terms of hardware, the data plane relies on:

- the **data interfaces** on the servers,
- redundant **Top-of-Rack (ToR) switches**,
- for multi-rack deployments, two additional families of switches:
    - **edge** switches, which provide the **junction between the OPCP internal network and your upstream network**: they carry the uplink configuration toward your *Customer Upstream Data Network*,
    - **spine** switches, which allow the platform to **scale beyond 3 racks** by aggregating the interconnection between the ToRs of several racks,
- **fibre uplinks** to your upstream network (*Customer Upstream Data Network*) in **40 GbE or 100 GbE**, depending on the optics validated by OVHcloud for your deployment.

The data plane is **driven end-to-end by OPCP** through OpenStack Neutron: creating a network, a subnet or a security rule from Horizon or the API automatically propagates to the underlying switches.

The out-of-band (OOB) administration network

The OOB network is the network **through which OPCP administers itself** and through which you (and, in Fully managed by OVHcloud mode, the OVHcloud teams) reach the management interfaces. It is **physically and logically separated** from the data plane. It carries:

- access to the **OpenStack APIs**, the **Horizon** interface, and the OPCP management interfaces (HTTPS),
- SSH access to the **OPCP Core Controllers** (`opcp-cli` , `opcp-diag`),
- the **inter-controller exchanges** that provide **control plane resiliency** in **STANDARD POD** and **FABRIC** modes (state synchronisation, quorum election, VIP failover between the three OPCP Core Controllers),
- the logs and metrics emitted from OPCP toward your internal services,
- the outbound flows from OPCP toward your infrastructure services (NTP, DNS, S3[1] for backups, LDAP, syslog).

In terms of hardware, the OOB network relies on:

- the **OPCP Core Controllers**, each equipped with an OOB uplink negotiating at **10 GbE** related to your upstream infrastructure, plus a 1 GbE BMC port,
- **uplinks** toward your upstream OOB network (*Customer Upstream OOB Network*).

> **Warning**
>
> In **STANDARD POD** and **FABRIC** modes, the three OPCP Core Controllers are reached through a shared virtual IP address, whose proper operation requires L2 reachability on a common untagged segment between the three OPCP Core Controllers. The availability of the control plane therefore depends directly on the availability of this network connection: a prolonged outage of the customer's OOB network can disrupt access to the administration of the platform. The **NANOPOD** mode is not concerned, since it only has one controller. Data exchanged between the OPCP Core Controllers travels through OPCP's own network. Only the administrators' access to the control plane can be affected by issues on the OOB network.

> **Warning**
>
> No direct communication is possible between the data plane and the OOB network from inside OPCP. Any cross-network communication must go through your upstream network, which leaves you in full control of the security rules between the two perimeters.

3. Deployment topologies

OPCP is delivered in one of three deployment modes, sized for different use cases. These modes differ mainly in the number of supported racks and the resiliency of the control plane. The presence or absence of **spine** and **edge** switches between the racks and your upstream network is a direct consequence of this.

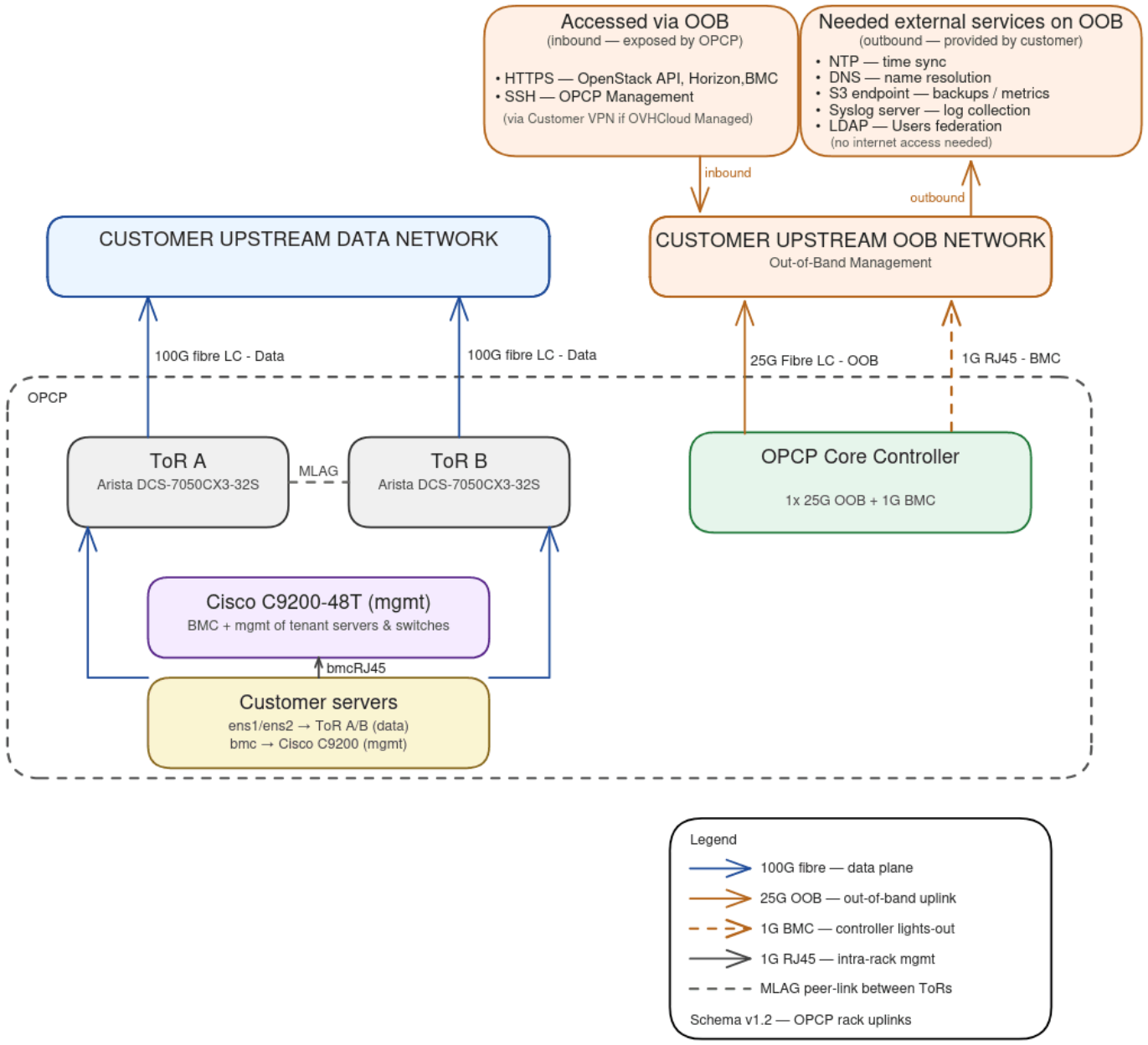
Mode	Use case	Control plane resiliency	Number of racks	Spine / Edge
NANOPOD	Demonstration, remote sites, non-resilient infrastructures (offices, edge)	Non-resilient (1 controller)	1	None
STANDARD POD	Compact resilient infrastructure	Resilient (3 controllers)	2 to 3	Edges only
FABRIC	Datacenter deployment with shared infrastructure	Resilient (3 controllers)	3 to 10 (and beyond with *megaspine*)	Spine + Edges

From a network integration standpoint, the key point to keep in mind is:

- in **NANOPOD**, the **data plane is configured directly on the Top-of-Rack (ToR) switches**: they are the ones connecting to your *Customer Upstream Data Network*;
- in **STANDARD POD** and **FABRIC**, the **data plane is configured on the edge switches**, which sit between the ToRs and your upstream network. The ToR configuration remains internal to OPCP.

The number of **OPCP Core Controllers** follows the deployment mode: **1 controller in NANOPOD**, **3 controllers in STANDARD POD and FABRIC**.

Single-controller configuration (NANOPOD)

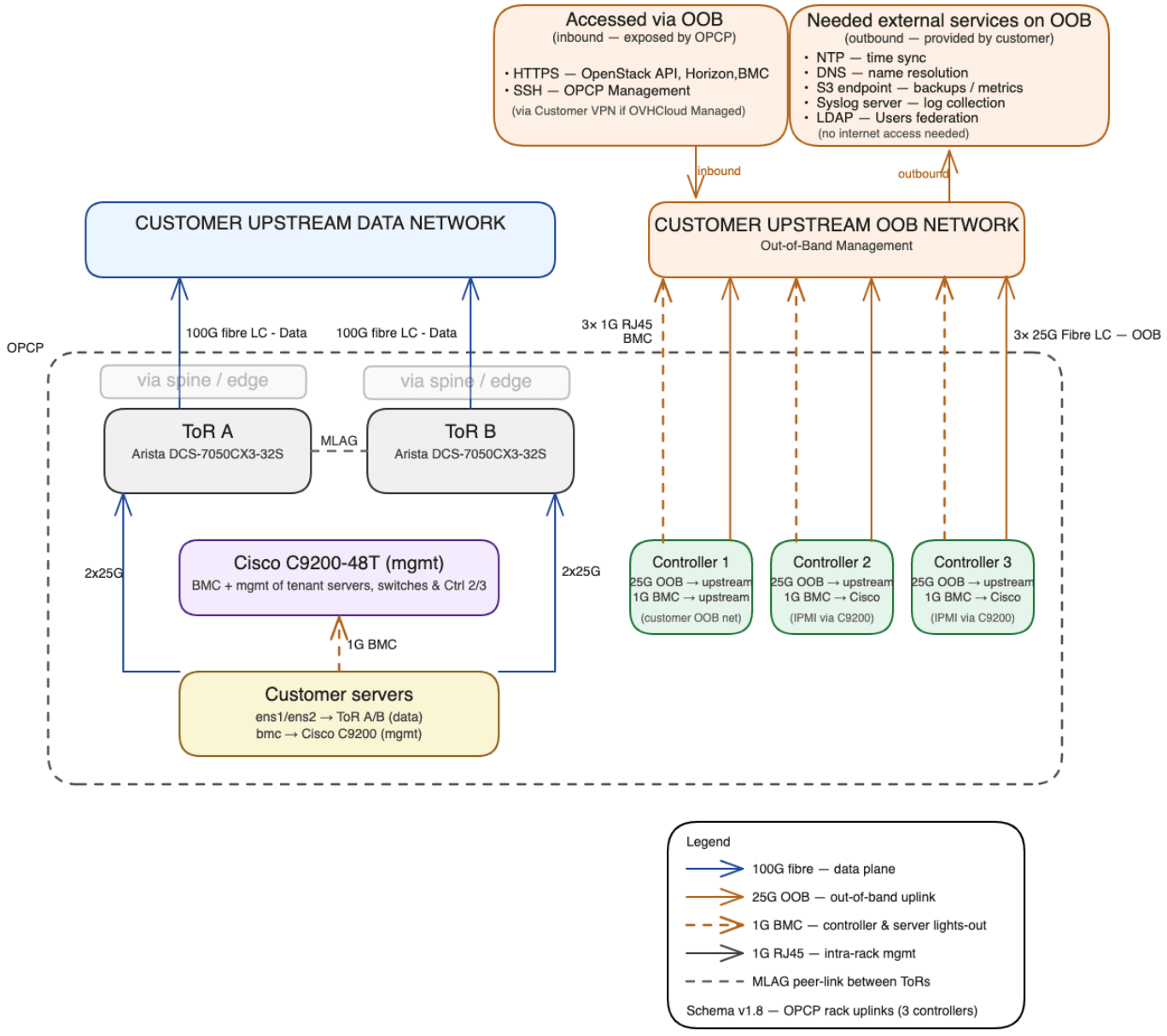


The key elements are:

- **Customer Servers**: the servers hosting the workloads. Their data interfaces are connected to both ToRs; their BMC is attached to the management switch.
- **ToR A / ToR B** (Arista DCS-7050CX3-32S): two redundant Top-of-Rack switches. In NANOPOD mode, **they carry the data plane directly and connect to your *Customer Upstream Data Network*** through fibre uplinks at **40 GbE or 100 GbE**, depending on the optics validated by OVHcloud.
- **Management switch (mgmt)**: dedicated switch that aggregates the server BMCs and the ToR management ports.
- A single **OPCP Core Controller**, connected through a **25 GbE or 10 GbE** OOB uplink to your *Customer Upstream OOB Network*, plus a 1 GbE BMC port for its own lights-out access.

> **Warning**
>
> With a single controller, **the control plane is not redundant**: a hardware failure or maintenance operation on this controller interrupts the administration of the platform (workloads already deployed continue to run on the servers). For any production environment requiring high availability of the control plane, choose the **STANDARD POD** or **FABRIC** modes.

Three-controller configuration (STANDARD POD and FABRIC)



Compared to NANOPOD mode, the differences are:

- **3 OPCP Core Controllers** instead of one, each connected to the upstream OOB network with a **25 GbE or 10 GbE** uplink and a 1 GbE BMC link. A **VIP** (*Virtual IP*) is shared between the three controllers: this VIP is used to reach the OpenStack APIs and Horizon, regardless of which controller is active.
- The BMCs of all three controllers, like those of the servers, are exposed on your upstream OOB network. This ensures lights-out access to every piece of equipment even if a component internal to OPCP fails.
- **Edge switches** are inserted between the ToRs and your upstream network. **The data plane is configured on these edges**, no longer on the ToRs. In **FABRIC** mode, **spine** switches are also present

to aggregate several racks; beyond ten racks, a **megaspine** layer can be added.

4. How OPCP integrates with your environment

The integration of OPCP relies on **two network perimeters** that you provide and keep under your control.

Data plane side (*Customer Upstream Data Network*)

This is your production network. OPCP connects to it through the fibre uplinks of the ToRs (in NANOPOD mode) or the edges (in STANDARD POD and FABRIC modes), at speeds of **40 GbE or 100 GbE** depending on the optics validated by OVHcloud.

The configuration of the network attachment (uplinks toward your upstream) is driven by OPCP through an API exposed by the platform. Depending on the deployment mode:

- in **NANOPOD**, the configuration is applied directly on the **ToRs**;
- in **STANDARD POD** and **FABRIC**, it is applied on the **edges**.

Beyond this attachment, the content and segmentation of the data plane remain **fully driven by OPCP** through OpenStack Neutron: each Neutron network is exposed on the uplinks using one of the three modes below.

Access mode

The uplink port is attached to **a single Neutron network** and presents traffic as classic **802.1Q** to your upstream equipment. This is the simplest mode and is suitable when an upstream port is dedicated to a single OPCP network.

If your instances push their own VLAN tags inside the Neutron network (hypervisor-level tagging, specific guest needs, etc.), those inner tags are transparently handled by OPCP and your upstream still only sees standard 802.1Q frames — there is nothing extra to configure on your side.

Trunk mode

The uplink port carries **several Neutron networks** over a single physical interface, each distinguished by its own VLAN ID. Because the VLAN tag set by OPCP is preserved end-to-end and never decapsulated, the framing your upstream sees depends on what happens **inside** each Neutron network:

- **802.1Q** — if your instances do not use VLANs inside the Neutron network. Your upstream equipment must be configured to recognise the VLAN IDs assigned by OPCP.
- **802.1ad (QinQ)** — if your instances push their own VLAN tags inside the Neutron network. The OPCP-assigned VLAN becomes the outer tag and the instance VLANs become inner tags. **Your upstream equipment must be able to handle double encapsulation.**

Both can coexist on the same trunk: each Neutron network is independent, so some can be plain 802.1Q while others are 802.1ad, depending on how each one is used.

Out-of-band side (*Customer Upstream OOB Network*)

This is the perimeter from which OPCP is administered. You need to plan for:

- a **dedicated subnet** for OPCP, with a default gateway. This subnet hosts the controllers (and their VIP in HA configuration),

- **DNS resolution**: a domain name you delegate to OPCP (for example `*.opcp01.example.com`), together with a *forwarder* from your internal DNS resolvers to the OPCP VIP,
- **filtering rules** allowing the flows described in the matrix below,
- **access to at least one NTP time source** from OPCP's administration network (mandatory) and, depending on the options you choose, to your other internal services (syslog, S3, LDAP).

5. Flow matrix to allow

All the flows below originate from or terminate on the **out-of-band (OOB) administration network** of OPCP. None of them uses the data plane.

Inbound flows (you → OPCP)

Source	Destination	Service	Protocol / Port
Administration workstations / OPCP users	OPCP controllers VIP	OpenStack API, Horizon, management interfaces	TCP/443
Administrators (Self-managed mode)	OPCP controllers	SSH (`opcp-cli`, `opcp-diag`)	TCP/22

Outbound flows (OPCP → you)

Source	Destination	Service	Protocol / Port	Mandatory
OPCP administration network	Your NTP servers	Time synchronisation	UDP/123	**Mandatory**
OPCP administration network	Your DNS resolvers	External name resolution	UDP/53	Optional
OPCP administration network	Your syslog servers	Log centralisation	UDP/514 or TCP/514	Recommended
OPCP administration network	Your S3 endpoint	Infrastructure backup	TCP/443	Recommended
OPCP administration network	Your S3 endpoint	Long-term metrics storage	TCP/443	Recommended
OPCP administration network	Your Active Directory / IdP	LDAP identity federation	TCP/636 (LDAPS)	Optional

> **Info**
>
> OPCP is designed to run in **air-gap** mode: none of these flows requires Internet access. The only expected destinations are services from your own information system.

6. The OVHcloud interconnection (Fully managed by OVHcloud mode)

If you have subscribed to the **Fully managed by OVHcloud** offering, OVHcloud needs to reach your OPCP administration network in order to provide remote operations and support. This interconnection relies on a **site-to-site IPsec tunnel** between an edge device on the OVHcloud side and an edge device on the customer side.

The default parameters proposed by OVHcloud are:

- **IKEv2 only** (IKEv1 is not supported),
- **PSK** (Pre-Shared Key) authentication,
- **AES-256** encryption,
- **SHA-256** hash,
- **Diffie-Hellman group 14**,
- **PFS** (Perfect Forward Secrecy) enabled.

> **Warning**
>
> OVHcloud strongly discourages lowering the encryption level below AES-256. If a constraint on your side prevents the use of these parameters, contact your OVHcloud point of contact to study the alternatives.

To prepare this interconnection, you must provide OVHcloud with:

- the **public IP address** of your IPsec endpoint,
- the **subnet** on your side to expose in the tunnel (typically the OPCP administration subnet),
- the agreed **PSK** (transmitted through a secure channel: check with your OVHcloud point of contact).

Once the tunnel is established, OVHcloud teams reach your OPCP **only** through this interconnection, using the same flows as those described in the matrix above. No direct access from the Internet is opened on your platform.

7. Addressing plan and naming

When preparing the deployment, you will provide OVHcloud with:

- the **subnet** allocated to the OPCP administration network and its **default gateway** (for example: subnet `172.30.1.0/24`, gateway `172.30.1.254`),
- the **hostnames and IP addresses** of the OPCP controllers, as well as the **VIP** shared in HA configuration (for example: `opcp-controller-0` / `172.30.1.10`, `opcp-controller-1` / `172.30.1.11`, `opcp-controller-2` / `172.30.1.12`, VIP `172.30.1.5`),
- the **domain name** you delegate to OPCP (for example: `opcp01.example.com`).

The addressing plan and naming convention are **entirely under your control**: OVHcloud does not impose any format. Refer to the OPCP technical prerequisites guide for the full list of elements to provide.

Go further

- Technical prerequisites for deployment
- Getting started with your OPCP

For training or technical assistance implementing our solutions, contact your sales representative or visit our Professional Services page to request a quote and have your project analyzed by our experts.

Join our community of users.

[1]: S3 is a trademark of Amazon Technologies, Inc. OVHcloud's service is not sponsored by, endorsed by, or otherwise affiliated with Amazon Technologies, Inc.

Technical prerequisites for deployment

Discover the list of configuration items you need to provide to OVHcloud to prepare and deploy your OPCP platform

Objective

Deploying **On-Prem Cloud Platform (OPCP)** at your site requires gathering a set of configuration items beforehand. Some are mandatory for the initial bootstrap and cannot be changed after installation; others are optional or can be adjusted later. Preparing this list upfront is the **main condition for a fast deployment with no back-and-forth**.

The aim of this guide is to present, for each prerequisite, what it is used for, how to express it, and whether it is mandatory or optional.

It is intended for both **architects** preparing the delivery and **administrators** who want to understand what was configured on their platform.

Requirements

- Having read the Network integration and platform connectivity guide, which describes the network architecture in which these prerequisites take place.
- Knowing the **OPCP subscription model** you have chosen (Self-managed or Fully managed by OVHcloud).
- Having an identified **OVHcloud point of contact** for the delivery, to whom you will transmit the gathered items.

Instructions

1. How to read this guide

For each prerequisite, you will find:

- a **description** explaining its role on the platform,
- whether it is **mandatory or optional**,
- whether it can be **modified after installation**,
- the **associated network flows** when the prerequisite implies a communication between OPCP and a service in your environment,
- a **generic example** to help you express the expected value.

> **Info**
>
> **Transmission of sensitive items.** Several prerequisites involve secrets: IPsec pre-shared key, S3[1] access keys, certificate private keys, and so on. These items **must not be transmitted through an insecure channel**. Agree with your OVHcloud point of contact on an appropriate transmission channel for each secret.

2. OVHcloud interconnection (managed mode only)

Field	Value
Mandatory	**Yes**, in Fully managed by OVHcloud mode
Modifiable after installation	Yes
Associated flow	Site-to-site IPsec tunnel between your endpoint and OVHcloud

If you have subscribed to the **Fully managed by OVHcloud** offering, an IPsec tunnel must be established between your site and OVHcloud to enable remote operations and support. You must provide:

- the **public IP address** of your IPsec endpoint,
- the **subnet** you expose on your side (typically the OPCP administration subnet),
- the agreed **pre-shared key (PSK)** for authentication, transmitted through a secure channel.

The default cryptographic parameters are **IKEv2 + PSK + AES-256 + SHA-256 + DH group 14 + PFS**. If your security policy requires different parameters, discuss this with your OVHcloud point of contact before deployment.

> **Warning**
> IKEv1 is not supported. Lowering the encryption level below AES-256 is discouraged: only consider it as a last resort.

In **Self-managed** mode, this prerequisite does not apply.

3. Administration network

Field	Value
Mandatory	**Yes**
Modifiable after installation	**No**
Associated flow	None (local OPCP configuration)

Definition of the management network on which OPCP will be positioned. You must provide:

- the **subnet** allocated to OPCP (for example: `172.30.1.0/24`),
- the **default gateway** address (for example: `172.30.1.254`).

This network will host the OPCP controllers and their VIP. It is also the starting point for the outbound flows toward your internal services (NTP, DNS, syslog, S3, LDAP).

> **Warning**
> These parameters are **frozen at deployment time**: the subnet and gateway cannot be changed after installation without a full reinstallation. Validate them carefully upfront.

4. OPCP environment variables

Field	Value
Mandatory	**Yes**
Modifiable after installation	**No**
Associated flow	None

OPCP uses several variables to uniquely identify the deployment. They appear in Netbox, in the logs, in the monitoring, and in several other components. You must provide a value for each of them:

Variable	Role	Example
`env`	Logical identifier of the OPCP environment	`opcp-prod-0`
`region`	Region or geographic zone code	`par`
`stage`	Lifecycle stage	`prod`, `staging`, `dev`
`org`	Owning organisation	`mycompany`
`site`	Physical site identifier	`dc1`
`location`	Precise location within the site	`R11-1`

> **Info**
>
> **OVHcloud does not impose any naming convention.** Adopt the nomenclature that matches your internal references. Choose it carefully: these values will be frozen and used by many components.

5. OPCP controller names and addresses

Field	Value
Mandatory	**Yes**
Modifiable after installation	**No**
Associated flow	None (local OPCP configuration)

For each **OPCP Core Controller** in your deployment, provide:

- the **hostname** (FQDN or short name depending on your convention),
- the **IP address** on the administration network.

In a 3-controller configuration, also provide the **VIP (Virtual IP)** shared between the three nodes, which will be the single entry point to the OpenStack APIs and Horizon.

Example in a 3-controller configuration:

Element	Name	IP
Controller 0	`opcp-controller-0`	`172.30.1.10`
Controller 1	`opcp-controller-1`	`172.30.1.11`
Controller 2	`opcp-controller-2`	`172.30.1.12`
VIP	`opcp.example.com`	`172.30.1.5`

6. Certificates

Field	Value
Mandatory	**Yes**
Modifiable after installation	Yes
Associated flow	None (local OPCP configuration)

OPCP must present valid TLS certificates for its interfaces (OpenStack API, Horizon, internal services). Three options are supported; you must choose one of them before deployment.

Option A — Recommended - Intermediate certificate authority provided by the customer

You provide an intermediate CA derived from your internal PKI. OPCP uses it to sign the service certificates. You must transmit:

- the **intermediate CA certificate**,
- the associated **private key** (transmitted through a secure channel agreed with your OVHcloud point of contact).

The rotation of certificates issued under this intermediate remains handled by CertManager on the OPCP side.

Option B — Self-signed certificate authority generated by OPCP

OPCP generates its own internal certificate authority and signs the required certificates. **You do not need to provide anything.** Rotation is handled automatically by CertManager.

This is the simplest option, suitable if your security policy accepts a CA internal to the OPCP perimeter. You will however need to distribute OPCP's root certificate to your clients in order to avoid security warnings.

Option C — Let's Encrypt

OPCP requests certificates automatically from Let's Encrypt. This option requires:

- a compatible **validation method** (HTTP-01 or DNS-01) reachable from OPCP,
- the **associated prerequisites** for this method (public resolution of the domain, outbound access to ACME servers, and so on).

Specify with your OVHcloud point of contact the method you have chosen and the configurations to put in place on your side.

7. NTP — Time synchronisation

Field	Value
Mandatory	**Yes**
Modifiable after installation	Yes
Associated flow	OPCP administration network → your NTP servers — UDP/123

OPCP needs a reliable time source for the correct operation of all of its components (log consistency, certificate validity, quorum election, and so on). Provide:

- one or more **IP addresses** of NTP servers, **or DNS names** if DNS resolution is configured.

Example:

- `172.30.1.200` / `ntp1.example.com`
- `172.30.1.201` / `ntp2.example.com`

Plan for opening the **UDP/123** flow from the OPCP administration network to your NTP servers.

8. DNS — Domain delegation to OPCP

Field	Value
Mandatory	**Yes**
Modifiable after installation	**No** (the DNS forwarder can be adapted as long as the FQDNs remain stable)
Associated flow	Your DNS resolvers → OPCP VIP — UDP/53 and TCP/53

OPCP exposes its interfaces through FQDNs under a domain that you delegate to it. You must provide:

- the **domain name** allocated to this OPCP platform (for example: `opcp01.example.com`).

On the DNS infrastructure side, you must create a **forwarder** from your internal resolvers to the OPCP VIP, so that any request for `*.opcp01.example.com` is resolved by OPCP.

9. DNS resolvers — External resolution

Field	Value
Mandatory	Optional
Modifiable after installation	Yes
Associated flow	OPCP administration network → your DNS resolvers — UDP/53

If OPCP needs to resolve **external** domain names (for example the FQDN of your S3 endpoint or your LDAP directory), provide the address of one or more DNS resolvers.

Example:

- `172.30.1.100`
- `172.30.1.101`

This prerequisite is only needed if you enable integrations relying on external FQDNs (S3 backup, long-term metrics, LDAP federation, and so on). For a strictly air-gapped platform configured by IP, it can be omitted.

10. Syslog — Log centralisation

Field	Value
Mandatory	Optional but **recommended**
Modifiable after installation	Yes
Associated flow	OPCP administration network → your syslog servers — UDP/514 or TCP/514

OPCP can forward its logs to a centralised syslog infrastructure for long-term retention and analysis. Provide:

- the **IP address** or FQDN of the syslog server (for example: `172.30.1.250`),
- the **listening port**,
- the **protocol**: TCP or UDP.

> **Info**
>
> Without an external syslog, OPCP keeps the logs **locally** with a default retention of **7 days** and a maximum volume of **50 GB**. Beyond that, the oldest logs are deleted. For any compliance requirement imposing longer retention, configure an external syslog.

11. Backup — S3 endpoint

Field	Value
Mandatory	Optional but **recommended**
Modifiable after installation	Yes
Associated flow	OPCP administration network → your S3 endpoint — TCP/443

OPCP can back up the infrastructure state (configurations, control plane state, metadata) to an S3-compatible endpoint that you provide. You must transmit:

- the **S3 endpoint** (for example: `s3.example.com:443`),
- the **Access Key**,
- the **Secret Key**,
- two **bucket names** dedicated to backups:
  - one for the **Velero** backups (persistent volumes — PV), for example: `opcp01-backup-pv-dc1`,
  - one for the **CNPG** backups (databases — DB), for example: `opcp01-backup-db-dc1`,
- the **S3 region name** (for example: `paris`).

The access keys must be transmitted through a secure channel agreed with your OVHcloud point of contact.

> **Warning**
>
> Without an external backup, you have no recovery mechanism in the event of a major incident on the platform. This option is strongly recommended for any production environment.

12. Long-term metrics storage — S3 endpoint

Field	Value
Mandatory	Optional but **recommended**
Modifiable after installation	Yes
Associated flow	OPCP administration network → your S3 endpoint — TCP/443

OPCP continuously collects metrics on the platform. To retain them beyond the local retention window, you can offload them to an S3 bucket. The elements to provide are the same as for the backup, but the bucket must be **separate**:

- **S3 endpoint**,
- **Access Key**,
- **Secret Key**,
- **bucket name** dedicated to metrics (for example: `opcp01-metrics-dc1`),
- **region name**.

You can reuse the same endpoint and the same S3 credentials as for the backup, provided you use a different bucket.

13. LDAP — Identity federation

Field	Value
Mandatory	Optional
Modifiable after installation	Yes
Associated flow	OPCP administration network → your directory — TCP/636 (LDAPS)

OPCP integrates Keycloak as an identity provider. If you wish to federate access with your corporate directory (Active Directory or another LDAP server), provide:

- the **IP addresses** or FQDNs of your LDAP servers (for example: `ldap.example.com`, `10.3.0.5`, `10.3.0.6`),
- the **listening port** (typically `636` for LDAPS).

Without federation, users are managed directly in the Keycloak embedded in OPCP.

14. SSH public key

Field	Value
Mandatory	Optional but **recommended**
Modifiable after installation	Yes
Associated flow	None (local OPCP configuration)

To access the OPCP controllers after the initial bootstrap (in particular to use `opcp-cli` and `opcp-diag`), provide one or more **SSH public keys** of the customer-side administrators.

In **Fully managed by OVHcloud** mode, this key gives the customer access to the administration tools in addition to the access of the OVHcloud teams.

> **Info**
>
> If no key is provided, a new SSH key pair will be generated during deployment. To keep control of your access from delivery onwards, it is preferable to provide the public key yourself.

Summary

Prerequisite	Status	Modifiable after installation
OVHcloud interconnection (IPsec)	Mandatory in managed mode	Yes
Administration network	Mandatory	**No**
OPCP environment variables	Mandatory	**No**
Controller names and addresses	Mandatory	**No**
Certificates	Mandatory	Yes
NTP	Mandatory	Yes
DNS (domain delegation)	Mandatory	**No**
DNS resolvers	Optional	Yes
Syslog	Optional but recommended	Yes
S3 backup	Optional but recommended	Yes
Long-term metrics S3 storage	Optional but recommended	Yes
LDAP	Optional	Yes
SSH public key	Optional but recommended	Yes

Pay particular attention to the prerequisites **not modifiable after installation**: they will shape your platform for the long term.

Go further

- Network integration and platform connectivity
- Getting started with your OPCP

For training or technical assistance implementing our solutions, contact your sales representative or visit our Professional Services page to request a quote and have your project analyzed by our experts.

Join our community of users.

[1]: S3 is a trademark of Amazon Technologies, Inc. OVHcloud's service is not sponsored by, endorsed by, or otherwise affiliated with Amazon Technologies, Inc.

Getting started with your OPCP

Find out how to manage and configure your On-Prem Cloud Platform

Objective

This guide has been designed to show you how to log in to the graphical interfaces of your **OPCP** as an administrator of this service.

Requirements

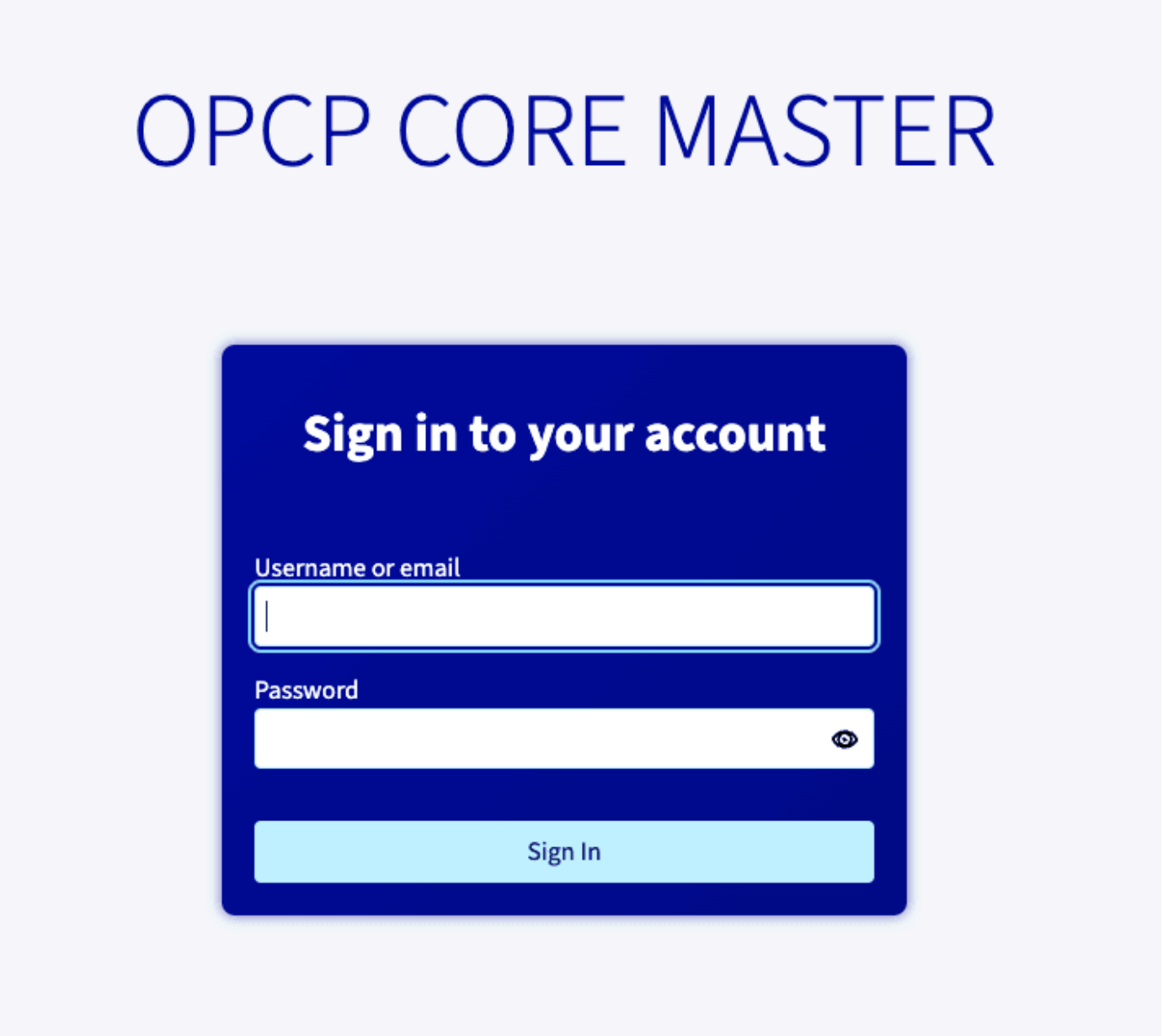
To follow this guide, you will need the following information

- The **url** address of the management interface defined for service delivery.
- Login and password details provided when the service was delivered.

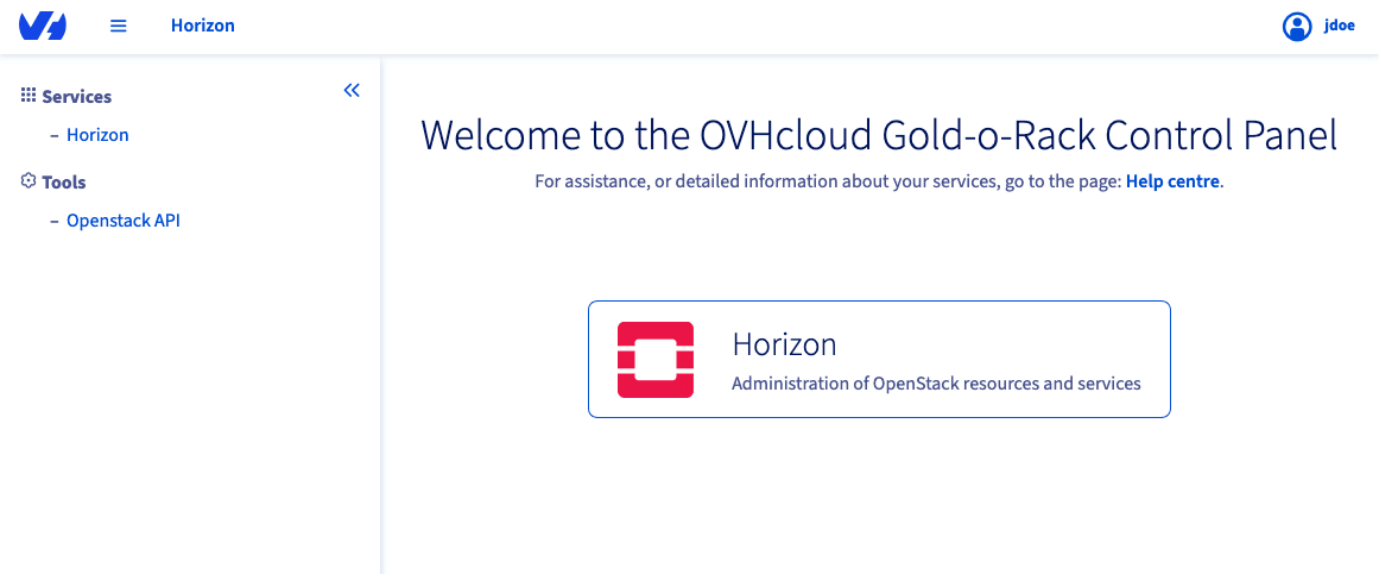
Instructions

User interface composition

The **url** address provided allows you to access the **OPCP** user interface.



Once you have used your login details to register, you will have access to the product dashboard.



Your interface allows you to access:

- The configuration of users within *Keycloak* via the IAM link under your login.
- The OpenStack management interface, *Horizon*. It is a graphical web interface for managing the entire OpenStack infrastructure. It allows the user to make use of the machine resources provided by the administrators. You can do this by creating, launching and stopping instances, configuring networks, and managing instance accessibility.

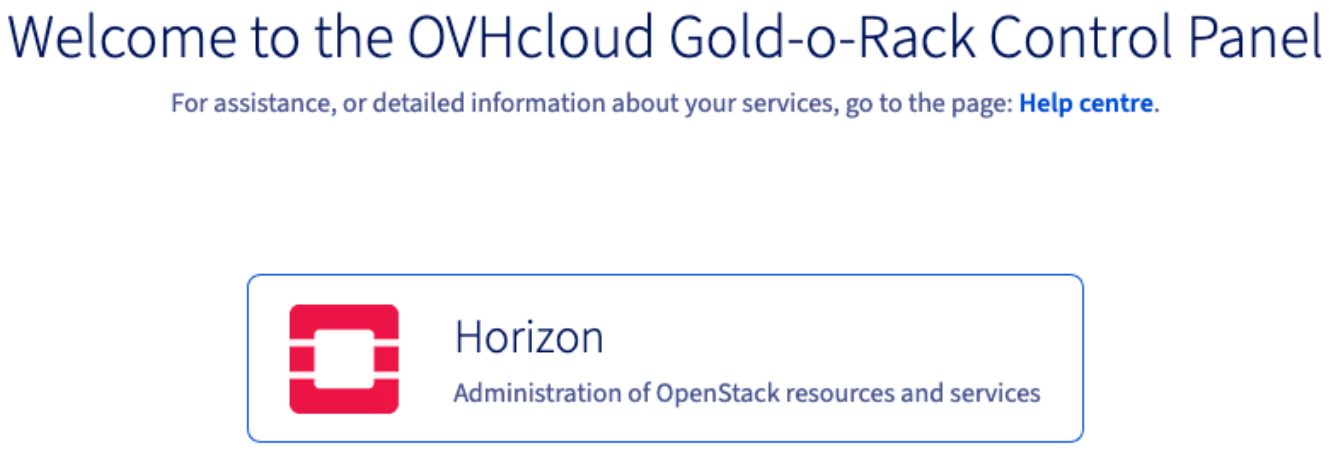
The **OPCP** administration interface also groups together access to various APIs such as Keystone (authentication and identity management), Glance (image management), Nova (computing service), Neutron (network management), Ironic (bare metal hardware management), which can be used within your automations.

Introduction to the OpenStack Horizon Interface

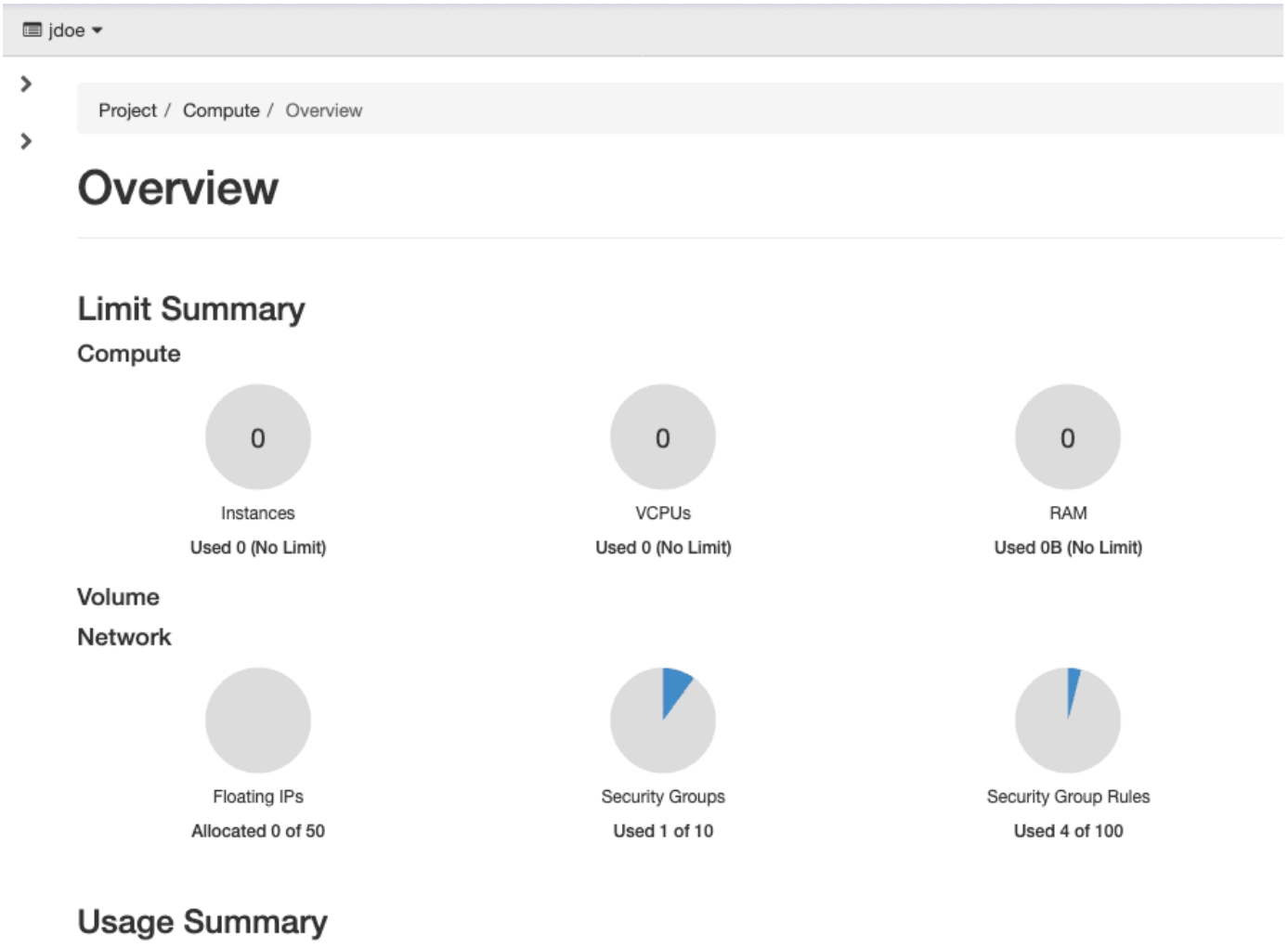
The OpenStack Horizon graphical interface allows you to perform different actions depending on their permissions and the project they belong to. Some of the main features available to an end user include instance management, network management, and resource tracking.

Access to the OpenStack Horizon administration interface

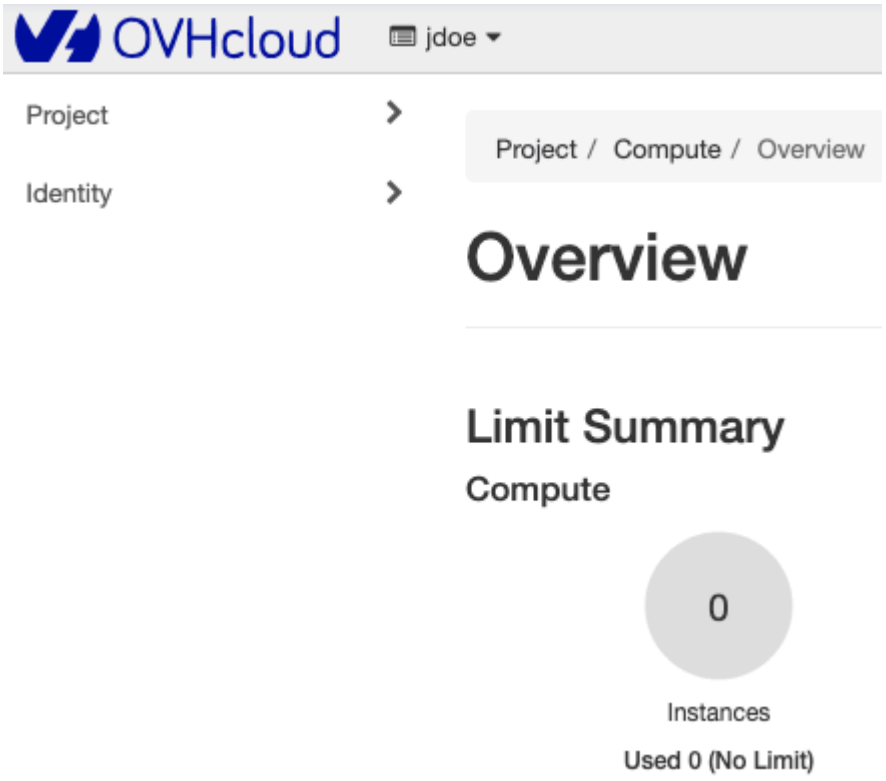
From the **OPCP** user interface, the OpenStack Horizon interface is accessible via the link in the dashboard.



Once you have logged in, the Horizon OpenStack interface looks like this:



The side menu to the left of the interface provides access to the various interface elements. There are two parent entries in this menu:

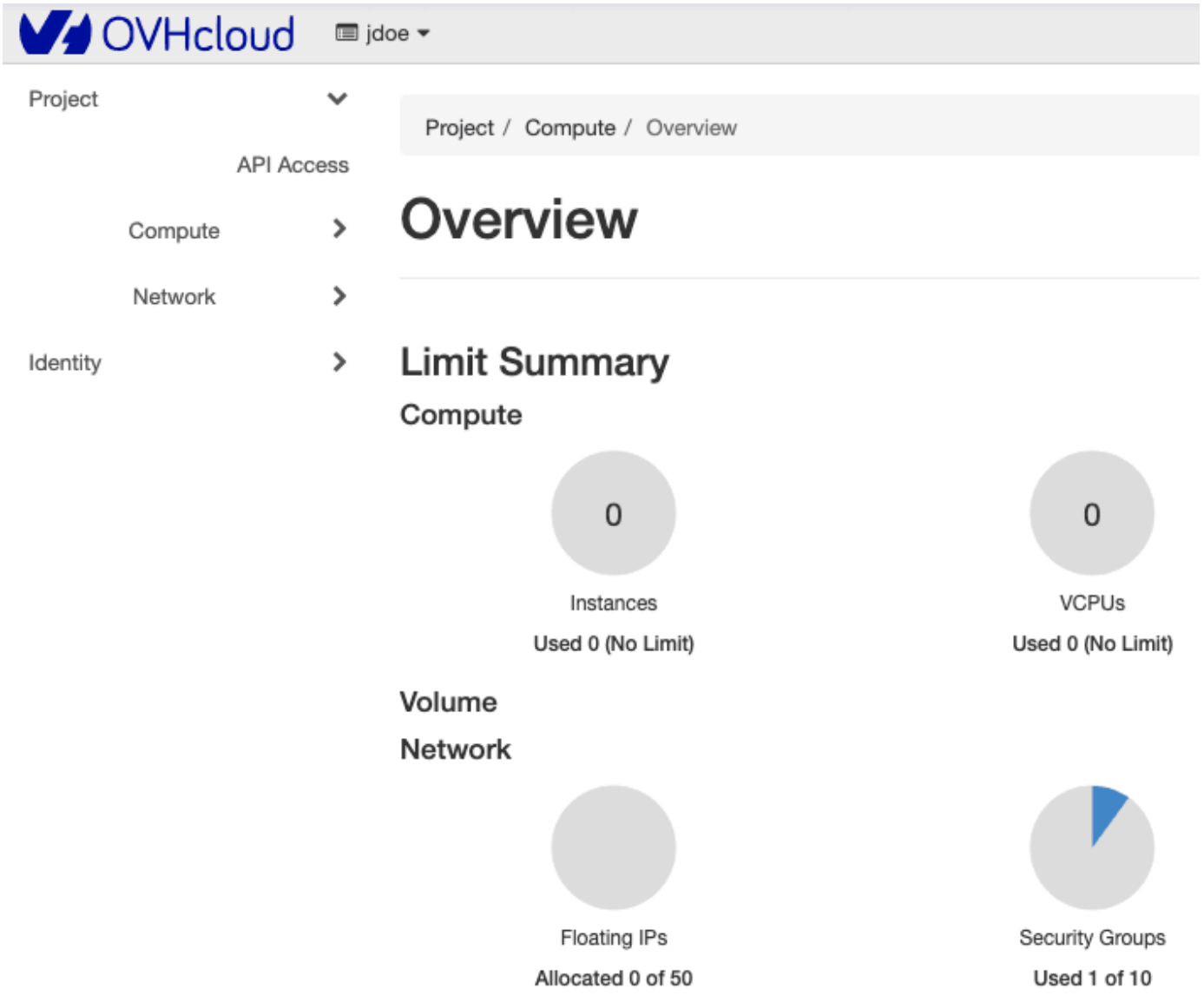


- **Project** which includes four elements: Overview, API Access, Compute and Network. These elements bring together all the management features of dedicated servers and their networks, within defined quota limits.
- **Identity** which includes Projects, Users, and Application ID elements that contain user management functionality.

Project view

The main *Project* element is made up of various sub-elements that provide access to all resource management functionality. The first sub-element, called *Overview*, provides a global overview of the project's allocated resource quotas, as well as a visual tracking of overall resource consumption.

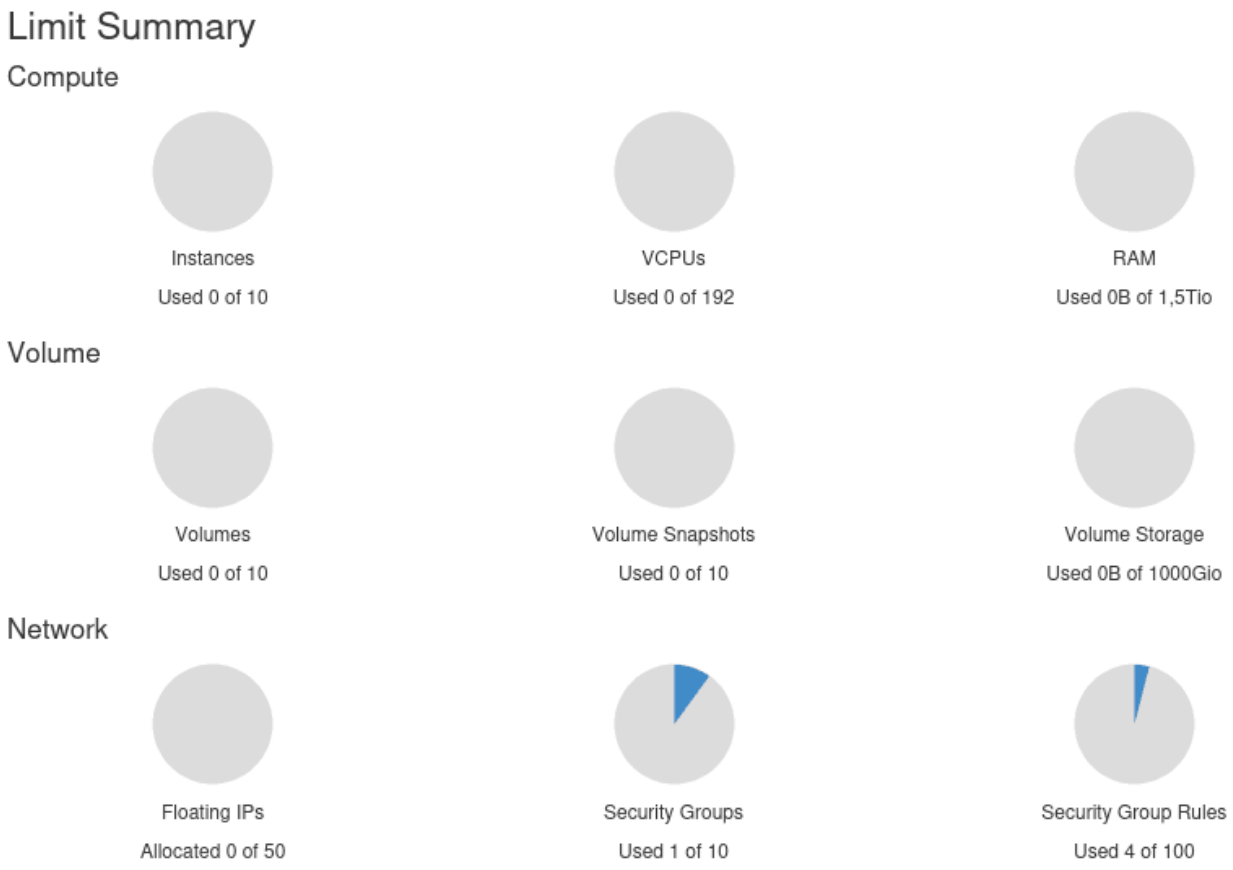
Overview section



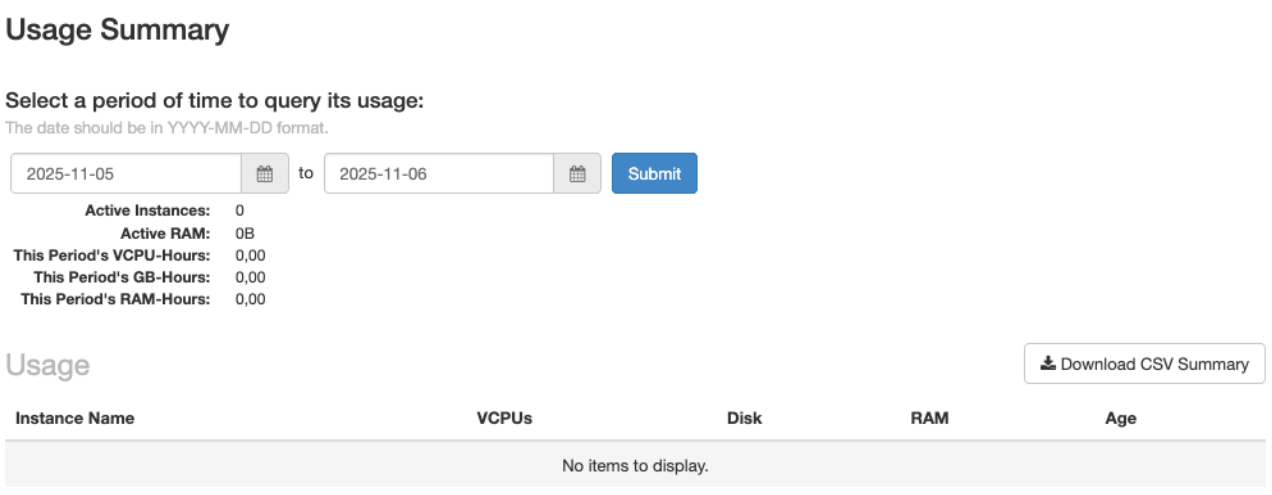
The *Overview* section is divided into two main parts:

- ***Limit Summary**: The quota limits assigned to the project for each resource type. You can also view resource consumption in relation to available capacity.

  Quotas are grouped into two categories, as shown in the image below:

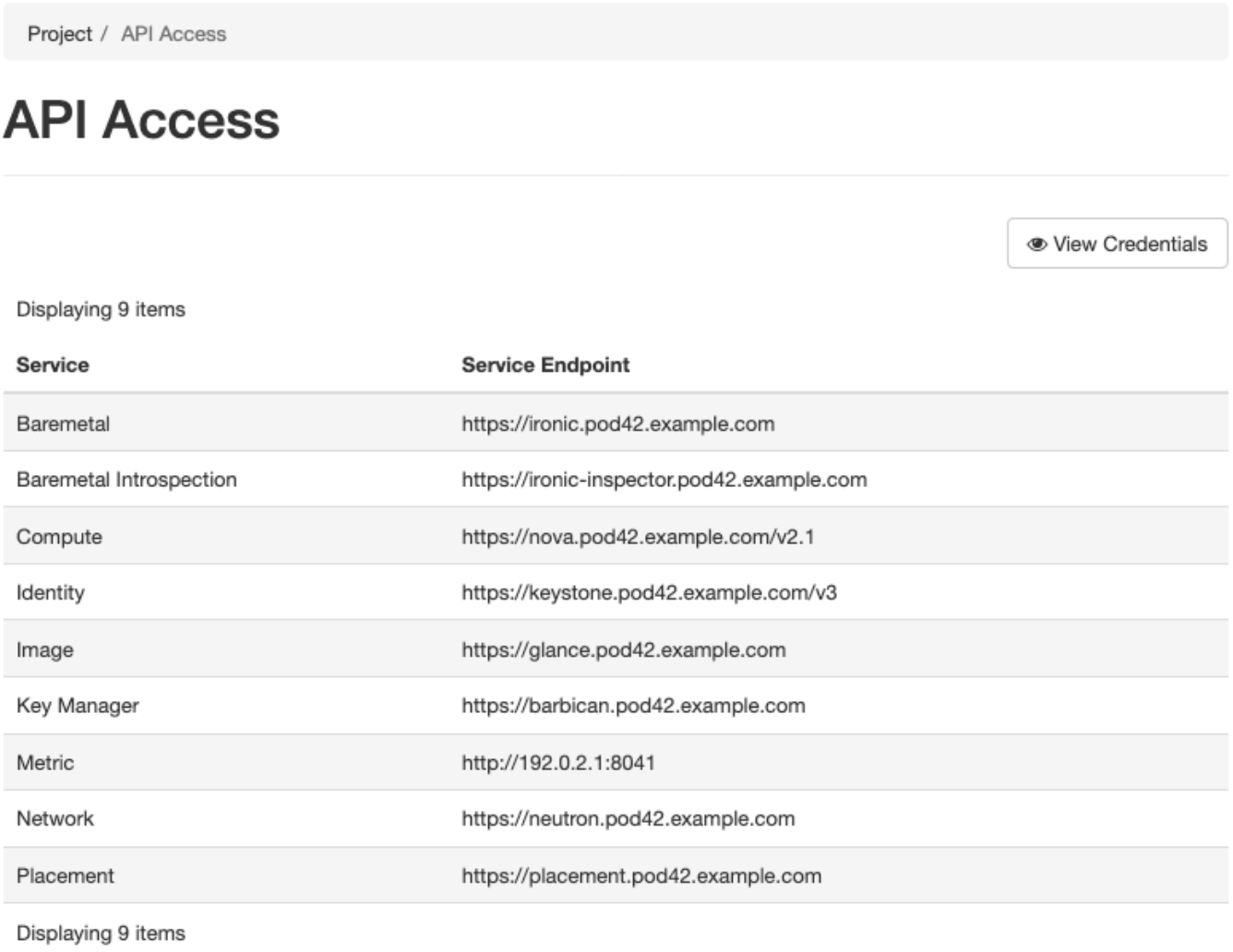


- *__Compute__ which includes instance limits, vCPUs and RAM.
    - __Network__ that monitors network resource quotas: Floating IPs, security groups, group security rules, networks, and ports.
    - __Usage Summary__: A history of resource usage over a period of time that allows you to see how resource usage has changed over time



API Access section

The **API-Access** tab groups together the 10 services available via the API, such as Bare-Metal, Compute, identity, image and network, as well as their access point URLs.



With these *Endpoints*, you can communicate directly with OpenStack components using API requests. You will need this information if you need to implement HTTP requests yourself using the OpenStack APIs.

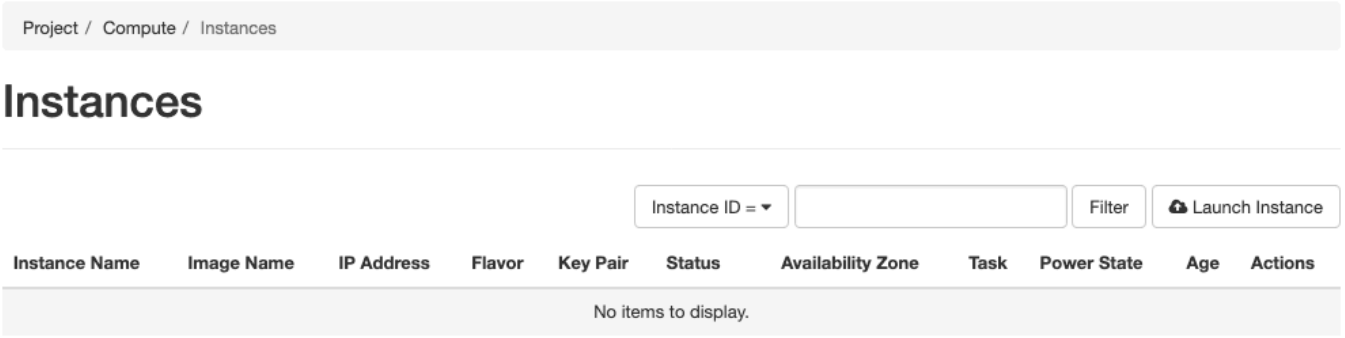
If you are using existing OpenStack integrations, they will retrieve this information the first time you log in to the OpenStack Keystone component. It is responsible for providing this information automatically.

Compute section

The **Compute** section provides features for configuring your product's dedicated servers. This section is divided into different sections:

Instances section

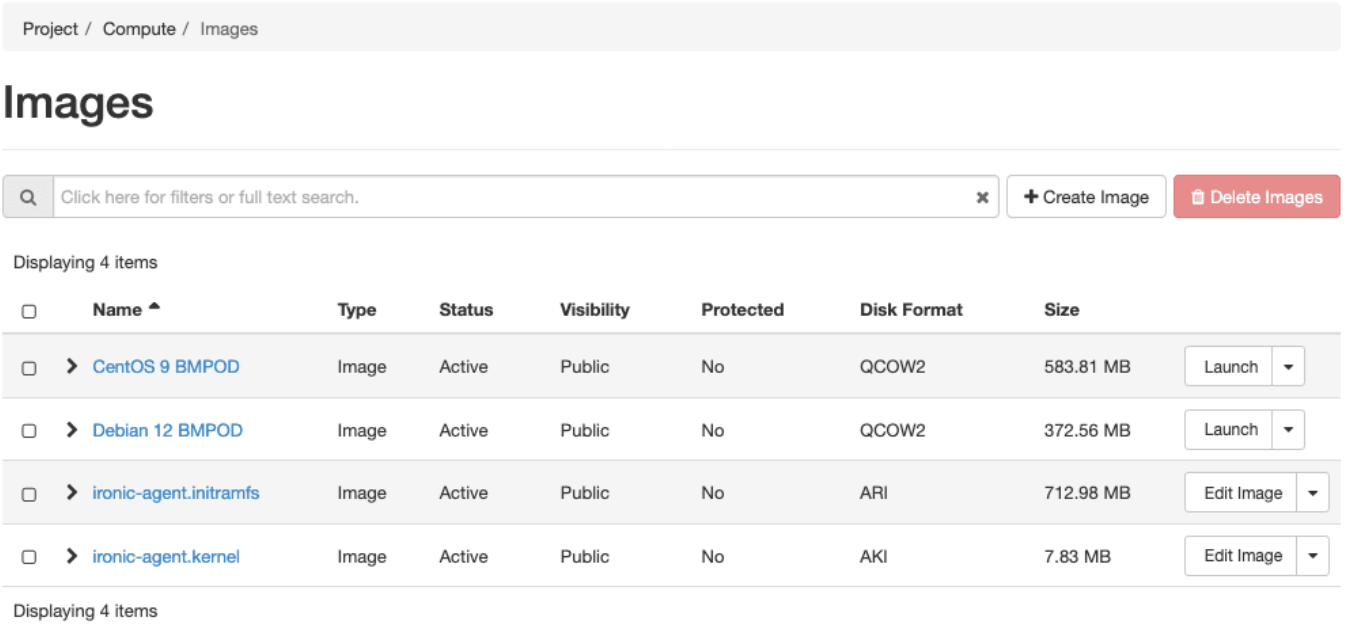
An interface for listing and managing dedicated servers that have already been configured. An *Instance* corresponds to a dedicated server.



Images section

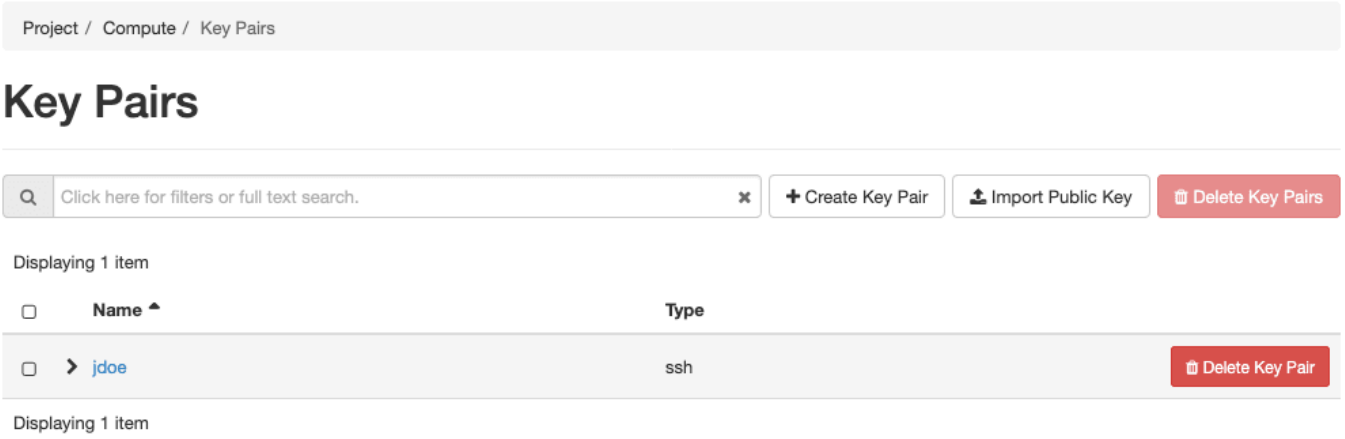
You can manage images of the OS available for creating instances. You can also download new images, or select from images already available, to set up instances. This way, you can generate your own images to manage additional operating systems.

> **Warning**
> The images must take into account the hardware drivers delivered in your product. Many images available for Glance are only available for virtual environments based on qemu or kvm drivers.



Key Pairs section

To authenticate yourself via SSH on your machines after installation, you will need to use asymmetric encryption keys. This interface allows you to import the public keys or create a keypair that will be deployed during the installation of the dedicated servers to ensure an SSH connection.



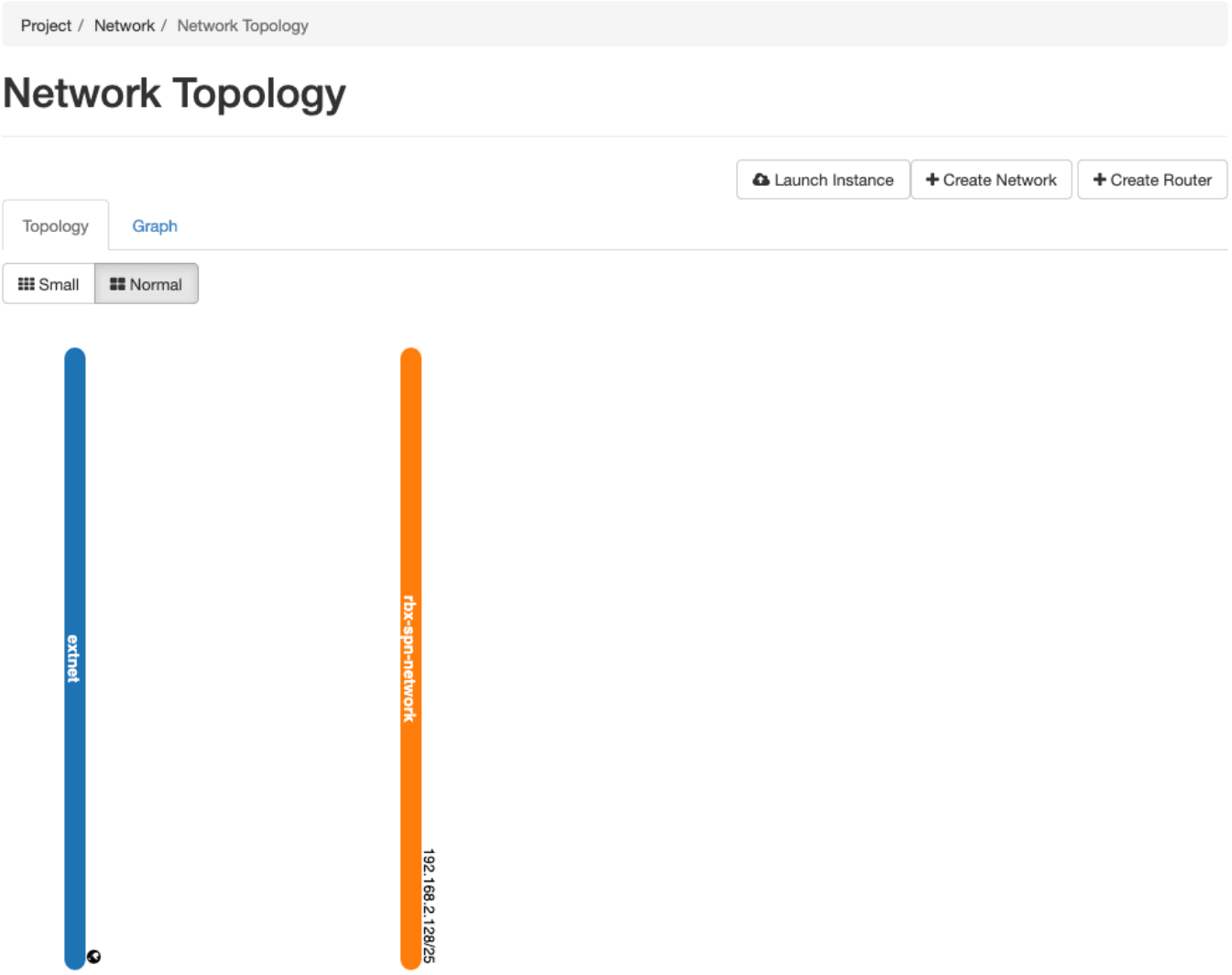
Network View

The Network View allows you to view and manage the networks of your **OPCP** service. This interface allows you to create shared or separate networks between your dedicated servers.

> **Info**
>
> Your entire network configuration is managed via this graphical interface or via the **OpenStack** APIs with the network component named **Neutron**. The switches in your infrastructure will be automatically configured using OpenStack information.

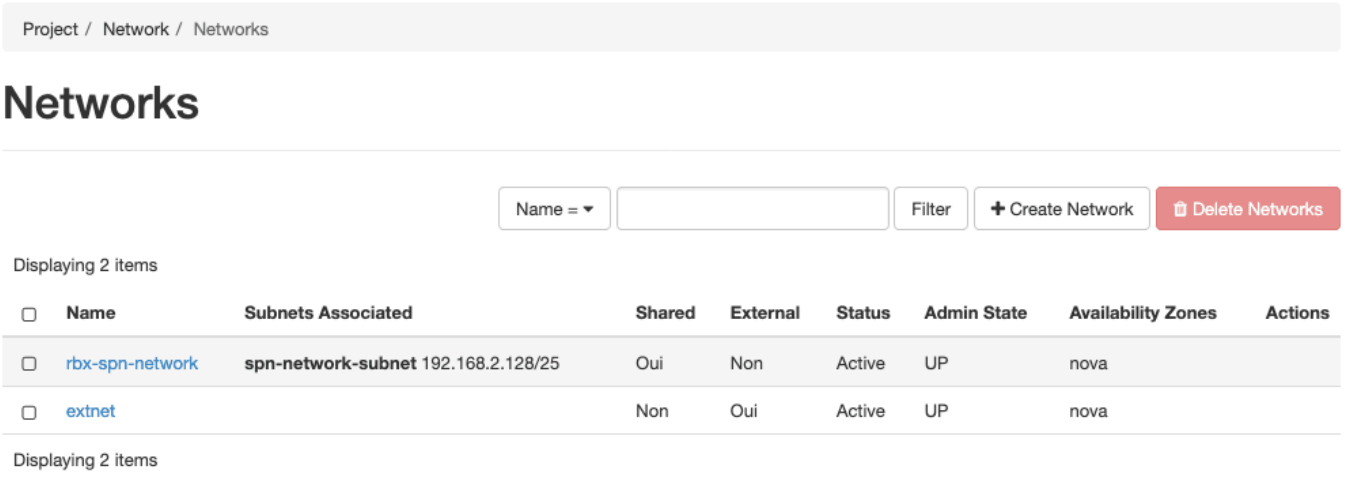
Network Topology section

This section shows you all the networks created on this OPCP via a vertical line of color. The squares correspond to services or dedicated servers connected to one or more of these networks.



Networks section

This section contains the list of networks available for dedicated servers on your **OPCP**.



To find out more about how networks work with OpenStack, we recommend reading the [OpenStack Networking] documentation.

Go further

If you need training or technical assistance for the implementation of our solutions, contact your sales representative or click this link to request a quote and have your project analyzed by our Professional Services team experts.

Join our community of users.

How to install a controller

Learn how to install an OPCP controller from the Debian image provided by OVHcloud, from preparing the media and selecting RAID 1 disks to configuring the access network.

Objective

This guide explains how to install an **OPCP** controller from the Debian-based installation image provided by OVHcloud. It covers preparing the installation media, cabling the server, selecting the RAID 1 disks, configuring the controller access network, and making the adjustments required after the first boot.

> **Info**
>
> In this guide, the **OOB** network refers to the controller access network. It is the server's first physical network interface, connected to your network.
>
> It is not the server's dedicated management port (IPMI, iDRAC, iLO, and so on), which should only be used to mount the ISO or open a remote console.

> **Warning**
>
> The installation process erases all data on the selected disks.
>
> OVHcloud provides you with services for which you are responsible. You must therefore ensure that they function correctly after the installation.

Requirements

- A physical server intended to host the OPCP controller
- Read-only access to the OPCP delivery S3[1] bucket, including the S3 endpoint, bucket name, region, and OPCP version to download
- A dedicated S3 access key and secret key, restricted to read-only access on that bucket and delivery prefix
- A USB key, virtual media device, or any other bootable installation media
- Access to the local console or to the server's remote management console (IPMI, iDRAC, iLO, and so on)
- At least 2 physical disks of comparable size for the RAID 1 installation
- One network cable connected to the server's first network interface and to your network
- The IP address, subnet mask, gateway, and DNS servers for the controller access network

Instructions

1. Download and verify the installation image

Before writing the image to your boot media, download the ISO and its SHA-256 checksum from the read-only S3 bucket provided to you.

> **Info**
>
> The published artifacts are stored in the delivery bucket under the `opcp-controller-debian-image/<version>/` prefix.
>
> Use a dedicated credential pair restricted to read-only access on that bucket and prefix. Load it only into your current shell or through your secret manager, then remove it after the download. Common base endpoints include `s3.sbg.io.cloud.ovh.net` and `s3.gra.io.cloud.ovh.net`.

Set your environment variables first:

```
export S3_ENDPOINT="<s3-endpoint>"
export S3_BUCKET="<s3-bucket>"
export S3_REGION="<s3-region>"
export OPCP_VERSION="<opcp-release-version>"
export S3_PREFIX="opcp-controller-debian-image/${OPCP_VERSION}"
export S3_ACCESS_KEY="<read-only-access-key>"
export S3_SECRET_KEY="<read-only-secret-key>"
```

Then download the files with `s3cmd` (the recommended method for authenticated S3 access):

```
sudo apt-get update
sudo apt-get install -y s3cmd

s3cfg=$(mktemp)
chmod 600 "$s3cfg"

cat >"$s3cfg" <<EOF
[default]
access_key = ${S3_ACCESS_KEY}
secret_key = ${S3_SECRET_KEY}
host_base = ${S3_ENDPOINT}
host_bucket = %(bucket).${S3_ENDPOINT}
bucket_location = ${S3_REGION}
EOF

for artifact in \
        live-image-amd64.hybrid.iso \
        live-image-amd64.hybrid.iso.sha256; do
        s3cmd -c "$s3cfg" get \
                "s3://${S3_BUCKET}/${S3_PREFIX}/${artifact}" \
                "${artifact}"
done

rm -f "$s3cfg"
```

If your environment already standardises downloads with `curl`, you can use a SigV4-signed HTTPS request instead:

```
download_with_curl() {
        local artifact="$1"
        local curl_config
        curl_config=$(mktemp)
        chmod 600 "$curl_config"

        cat >"$curl_config" <<EOF
url = "https://${S3_BUCKET}.${S3_ENDPOINT}/${S3_PREFIX}/${artifact}"
user = "${S3_ACCESS_KEY}:${S3_SECRET_KEY}"
aws-sigv4 = "aws:amz:${S3_REGION}:s3"
output = "${artifact}"
fail
silent
show-error
EOF

        curl --config "$curl_config"
        rm -f "$curl_config"
}

for artifact in \
        live-image-amd64.hybrid.iso \
        live-image-amd64.hybrid.iso.sha256; do
        download_with_curl "$artifact"
done
```

If your `curl` build does not support SigV4, use the `s3cmd` method.

Then verify the checksum before writing the ISO:

```
sha256sum -c live-image-amd64.hybrid.iso.sha256

unset S3_ACCESS_KEY S3_SECRET_KEY

sudo dd if=live-image-amd64.hybrid.iso of=/dev/sdX bs=4M status=progress oflag=sync
```

Replace `/dev/sdX` with the device corresponding to your installation media.

2. Cable the server

Before you start the installation:

- connect the server's first network interface to your network.
- if the server has several network interfaces, use the first one so that it is easier to identify during the installation, and do not connect any other one.

3. Boot the installer

Boot the server from the installation media, then follow the interactive installer.

In the GRUB menu, select the `Auto Install` entry to start the installation.

During the installation, the image:

- detects the available physical disks
- prompts you to select the disks to use for RAID 1

- configures the system automatically on the selected disks
- prompts you to choose the network interface used for controller access
- enables `root` SSH access on the controller at first boot

4. Select the installation disks

When prompted by the installer, select at least 2 physical disks for the controller.

Use identical disks whenever possible, or disks that are as close as possible in size and performance, so that the RAID 1 configuration can be created correctly.

The installer automatically creates the RAID 1 configuration and the system LVM layout on the selected disks.

5. Configure the controller access network

When the installer shows the list of physical network interfaces:

1. Select the interface connected to your network.
2. Choose `Static` to enter the configuration manually.
3. In static mode, enter the IP address, subnet mask, gateway, and DNS servers requested by the installer.
4. Confirm the summary before the network configuration is written.

At the end of the base installation, this configuration is applied to the installed system to allow initial SSH access to the controller.

6. Verify access after first boot

After the server restarts:

1. Connect from the network linked to the server's first network interface.
2. Verify that the controller IP address responds over SSH.
3. Verify that the controller can reach the network resources it needs in your environment.

If the interface or network settings are incorrect, restart the installation and select the expected values.

7. Adjust the logical volume sizes

After the first boot, log in to the controller and adjust the logical volume sizes according to the actual disk capacity and your needs.

> **Warning**
> The sizes below are example targets for 900 GB disks. Check the used and available space before reducing any volume, then adapt the values to your disk capacity.

Start by reducing the `spare` volume, then reallocate the released space to the volumes you actually use:

```
lvreduce -r -L 100G -v /dev/mapper/vg-spare
lvresize -r -L100G -v /dev/mapper/vg-home
lvresize -r -L100G -v /dev/mapper/vg-root
lvresize -r -L30G -v /dev/mapper/vg-tmp
lvresize -r -L200G -v /dev/mapper/vg-var
```

Go further

If you need training or technical assistance implementing our solutions, contact your sales representative or visit our Professional Services page to get a quote and request a custom analysis of your project from our Professional Services team.

Join our community of users.

[1]: S3 is a trademark of Amazon Technologies, Inc. OVHcloud's service is not sponsored by, endorsed by, or otherwise affiliated with Amazon Technologies, Inc.

Lifecycle of an OPCP Node

Discover the lifecycle of an OPCP node and its different statuses

Objective

A node in OpenStack represents the configuration of a physical server in the OPCP rack. It must be distinguished from instances, which represent the operating system running on a node.

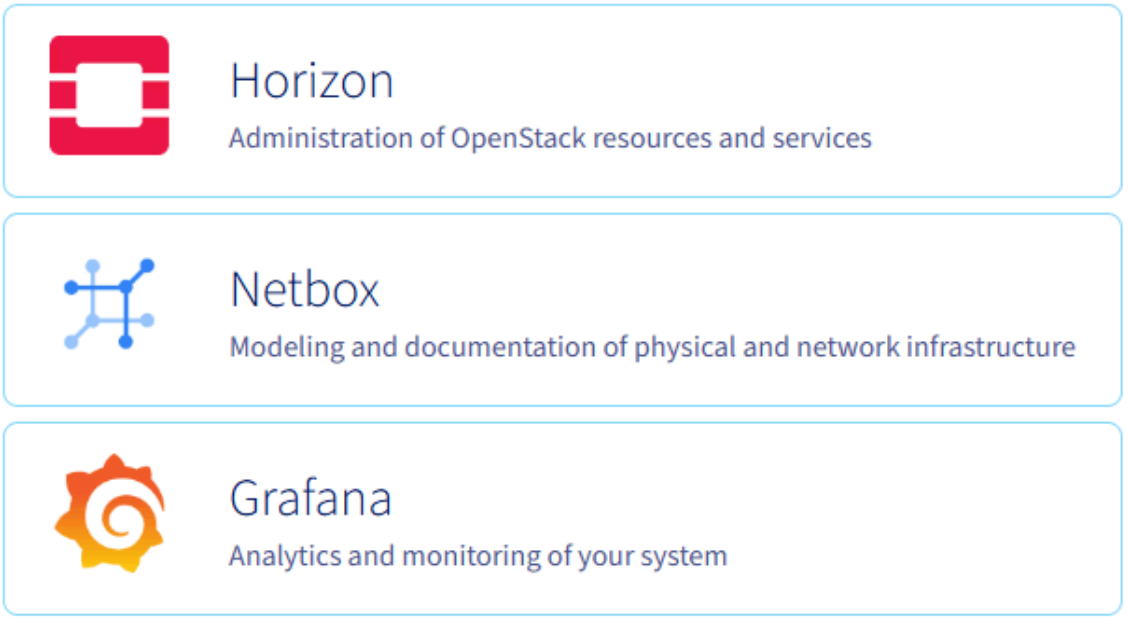
This guide details the different statuses of a node in an OPCP rack and how to modify them.

Requirements

- Have an active OPCP service.
- Have a user account with admin rights to log in to Horizon on the OPCP offering.
- (Optional) Have access to the OpenStack APIs for your project.
- (Optional) Have installed the Ironic client.

Instructions

Log in to the Horizon interface of your OPCP on the admin project.

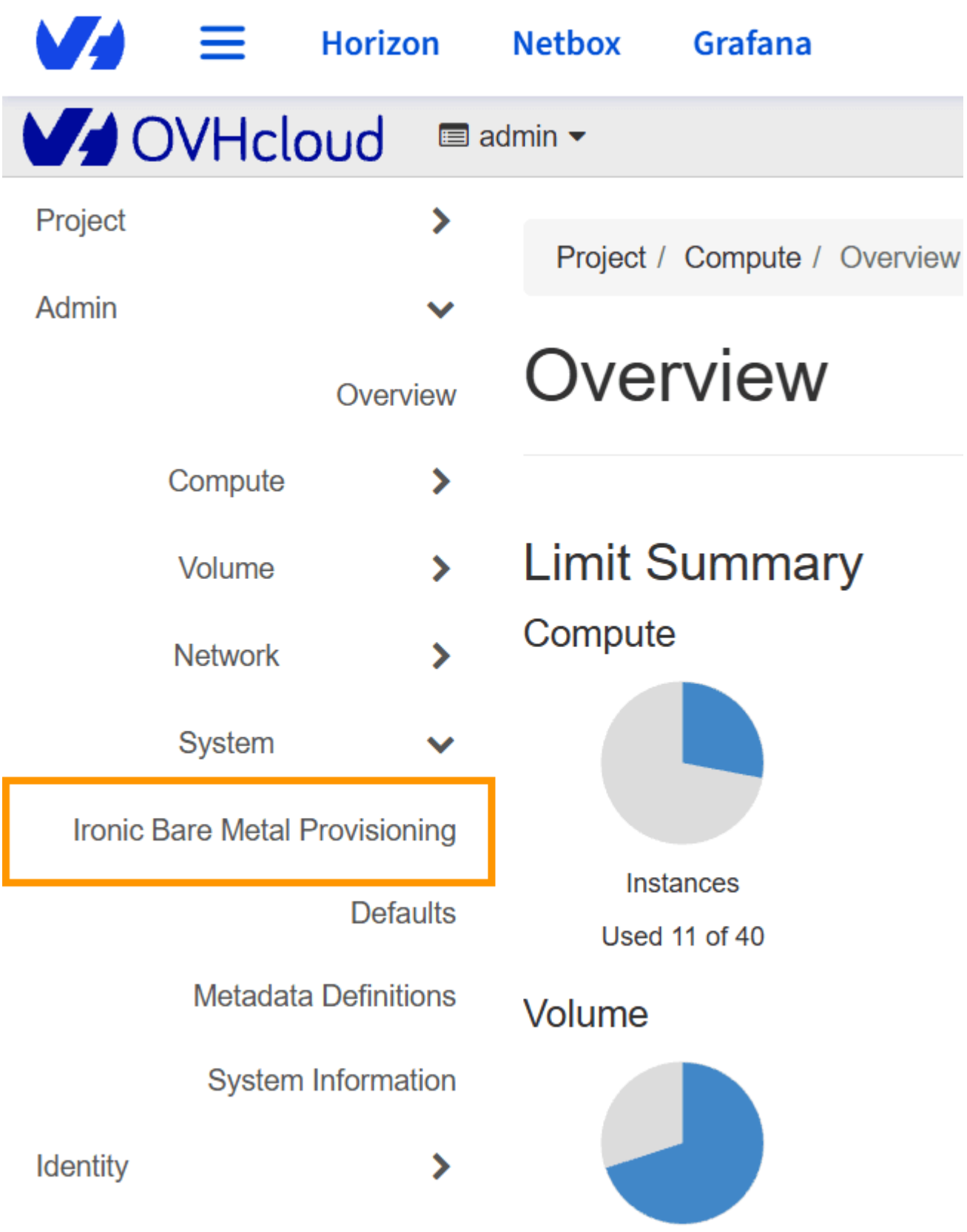


If you want to follow the OpenStack API section, you will need to install the Ironic packages on your environment:

```
pip install python-ironicclient
```

Check the status of a node

You can check the status of a node directly from Horizon via the `Admin` > `System` > `Ironic Bare Metal Provisioning` tab:



You will find the list of your nodes and their various statuses.

From the OpenStack APIs, you can retrieve the same list using the following command:

```
baremetal node list
```

You can also check the status of a specific node:

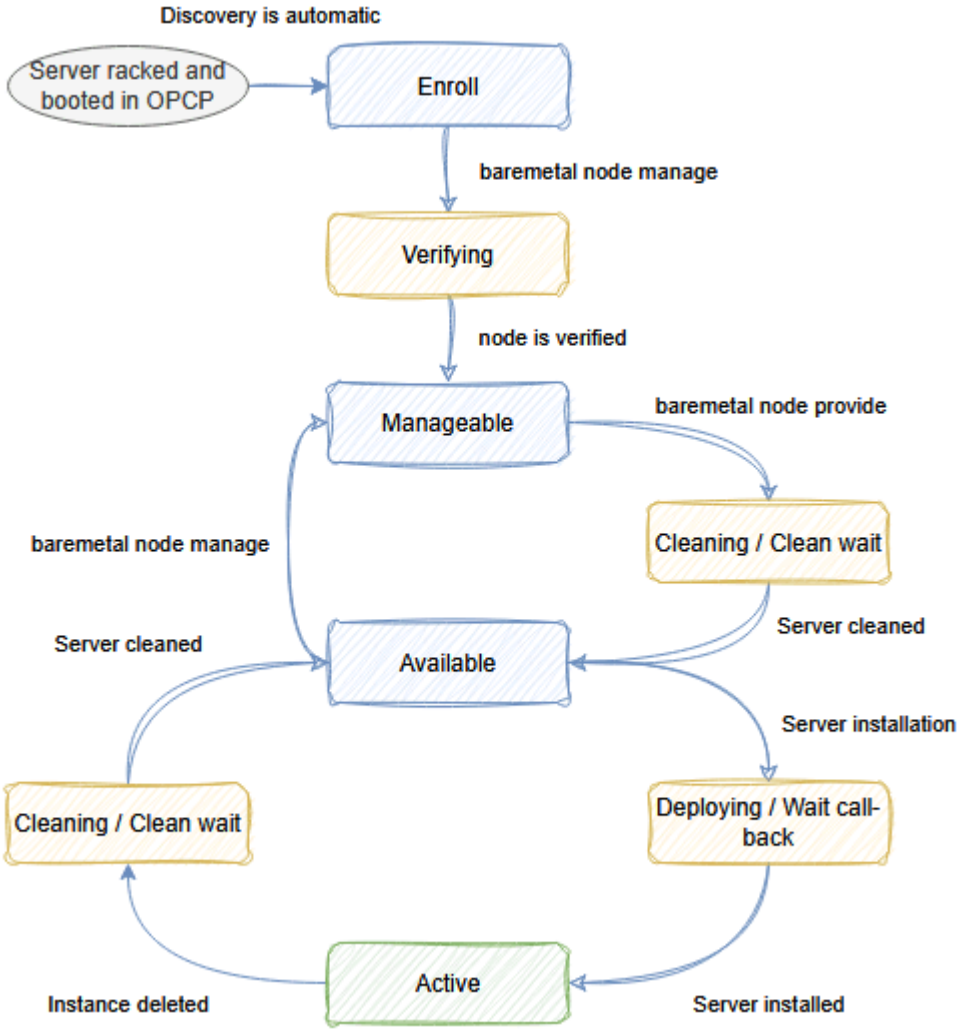
```
baremetal node show $BAREMETAL_NODE_ID
```

Possible statuses

Status	Description
Enroll	First state of the node when it has been automatically discovered by OPCP. The server has not yet been validated and must be manually transitioned to `Manageable`.
Manageable	The node has been verified and is managed by Ironic, but it is not yet installable. The node must be moved to the `Available` state before it can be deployed.
Available	The node is available and can be installed.
Active	The node is installed and has an active instance on it.
Verifying	Transitional state when a node moves from `Enroll` to `Manageable`. Ironic verifies it can manage the node using the drivers and hardware properties configured during control-plane discovery.
Cleaning / Clean-wait	Transitional state when an instance is deleted or when leaving the `Manageable` state before becoming `Available` again. Disks are wiped during this step.
Deploying / Wait call-back	Transitional state when the node is being deployed.

You can find detailed explanations for the different statuses in the official OpenStack documentation.

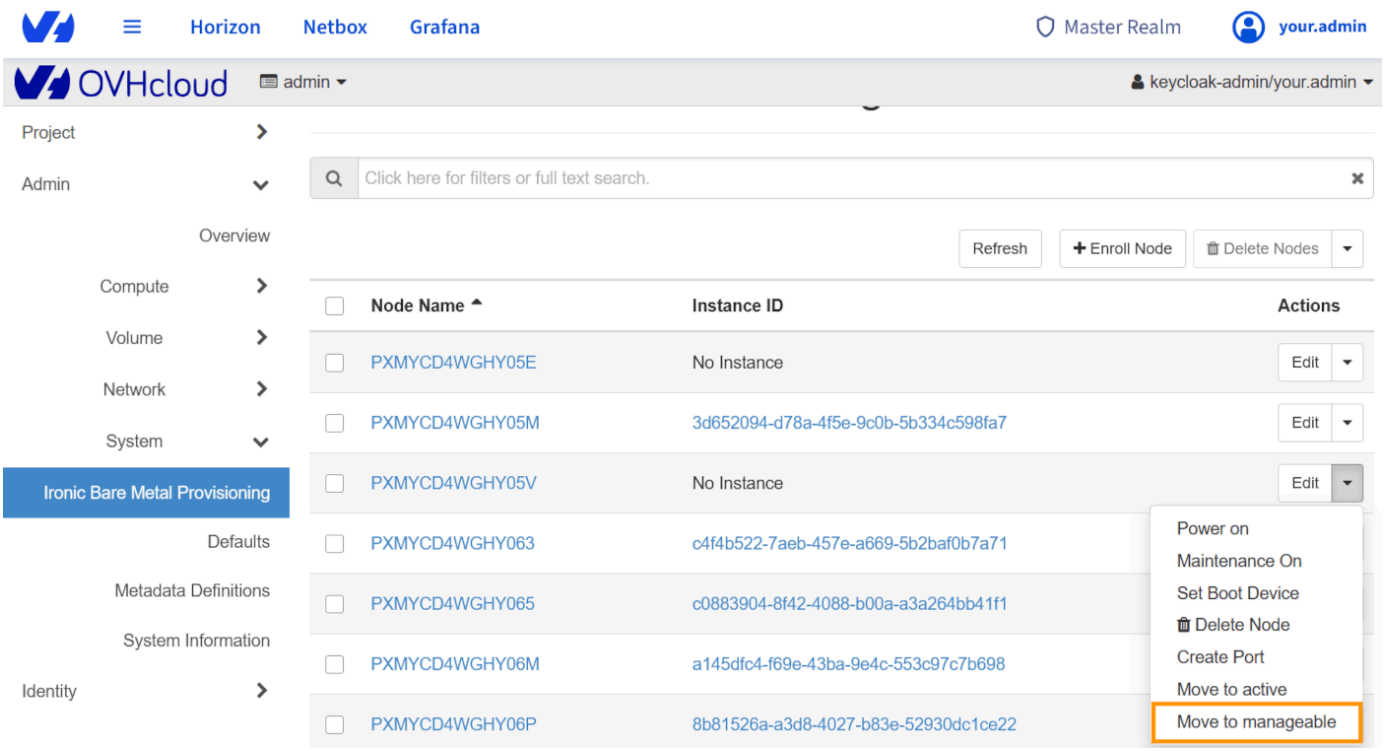
Node lifecycle



When a node is installed and booted in an OPCP rack, its discovery is automatically performed by the control plane. At this moment, the node retrieves its properties and **traits** based on its hardware profile.

Once the node is in the `Enroll` state, you can change its state so that it is managed by Ironic.

From the Horizon interface:

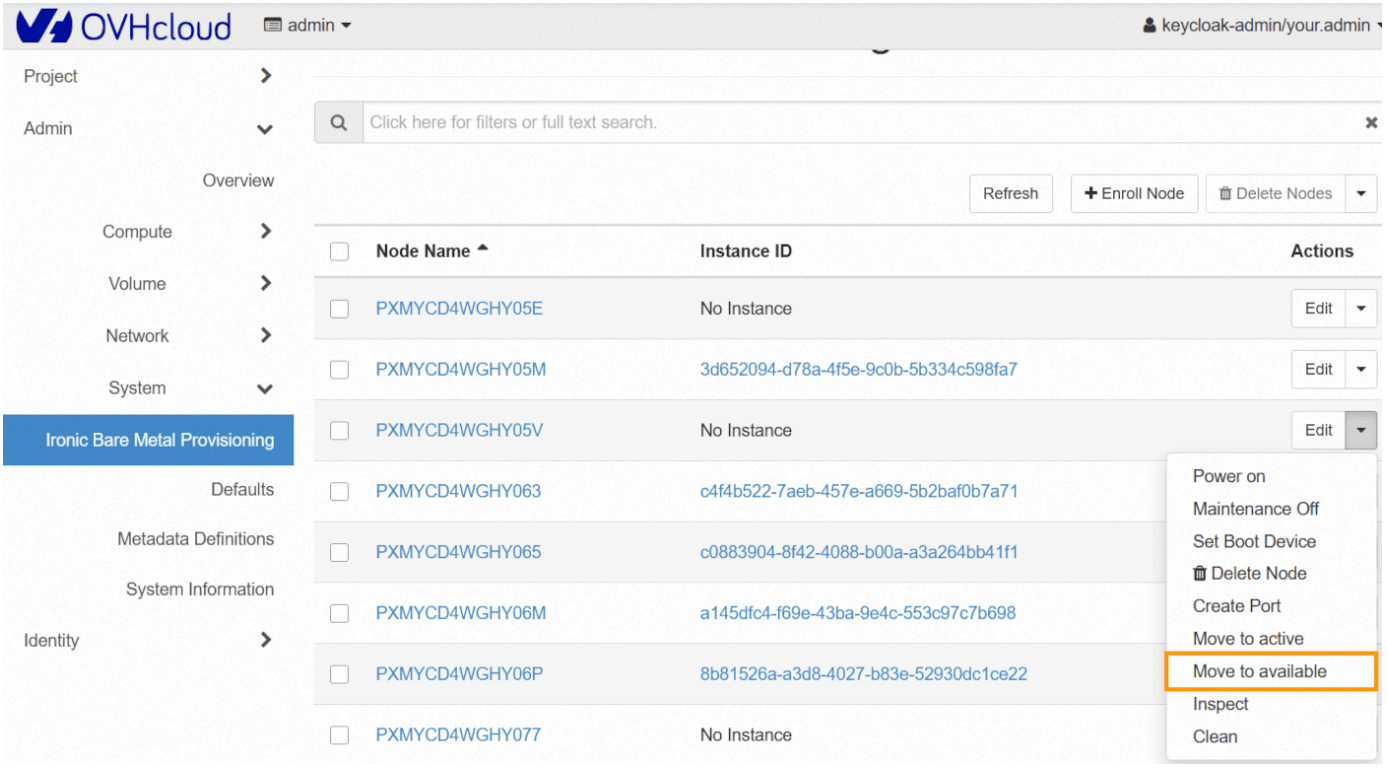


From the OpenStack APIs:

```
baremetal node manage $BAREMETAL_NODE_ID
```

To make the node available for installation, it must then be transitioned to the `Available` state:

From the Horizon interface:



From the OpenStack APIs:

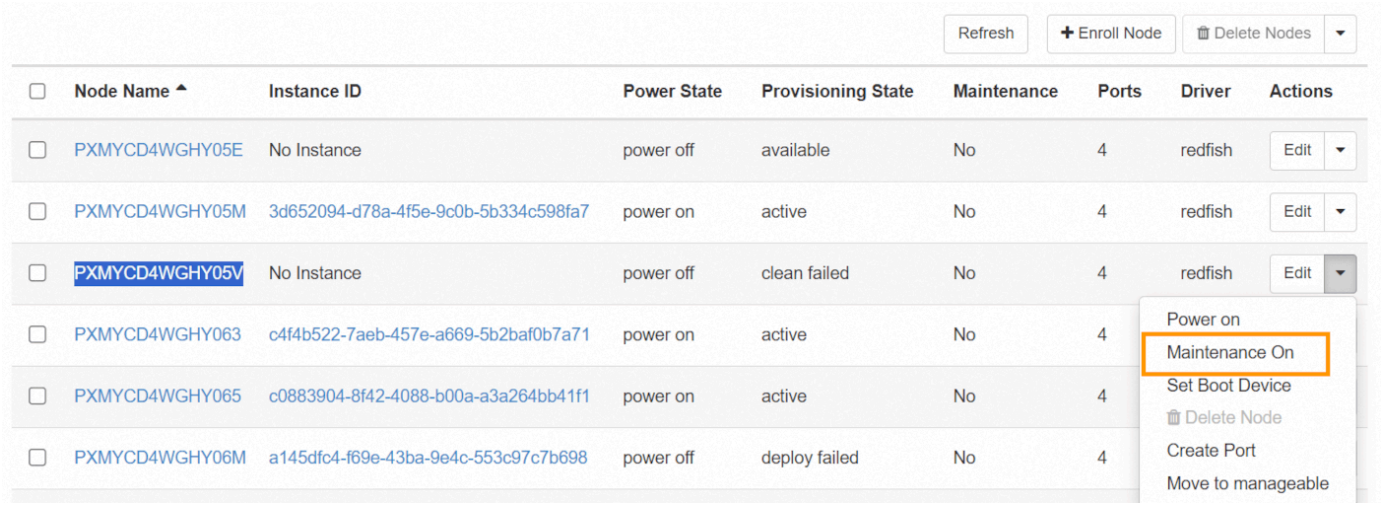
```
baremetal node provide $BAREMETAL_NODE_ID
```

The node then transitions to the `Cleaning` state before reaching `Available`, making it deployable by the various projects in your OPCP environment.

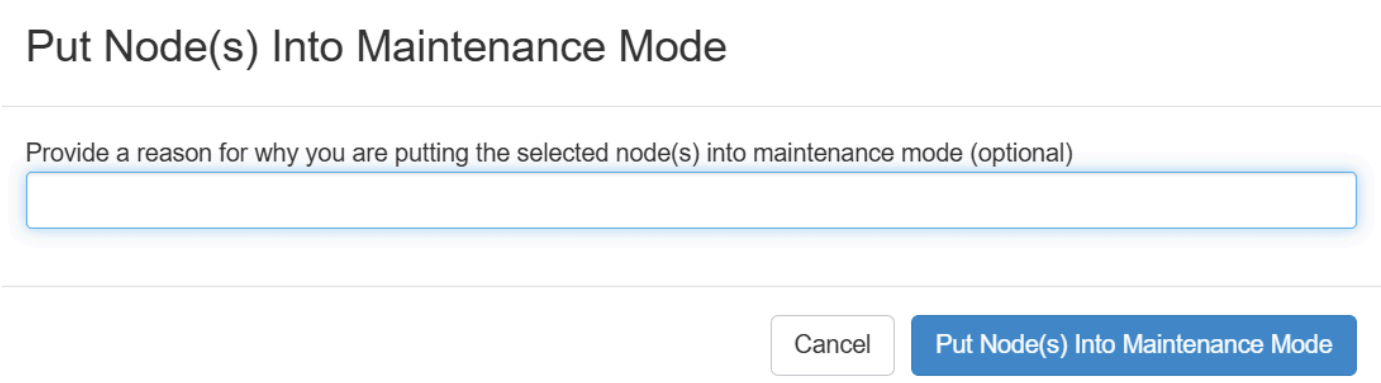
Maintenance mode

This mode can be enabled to ensure a node cannot be used for installation, even if it is in the `Available` state.

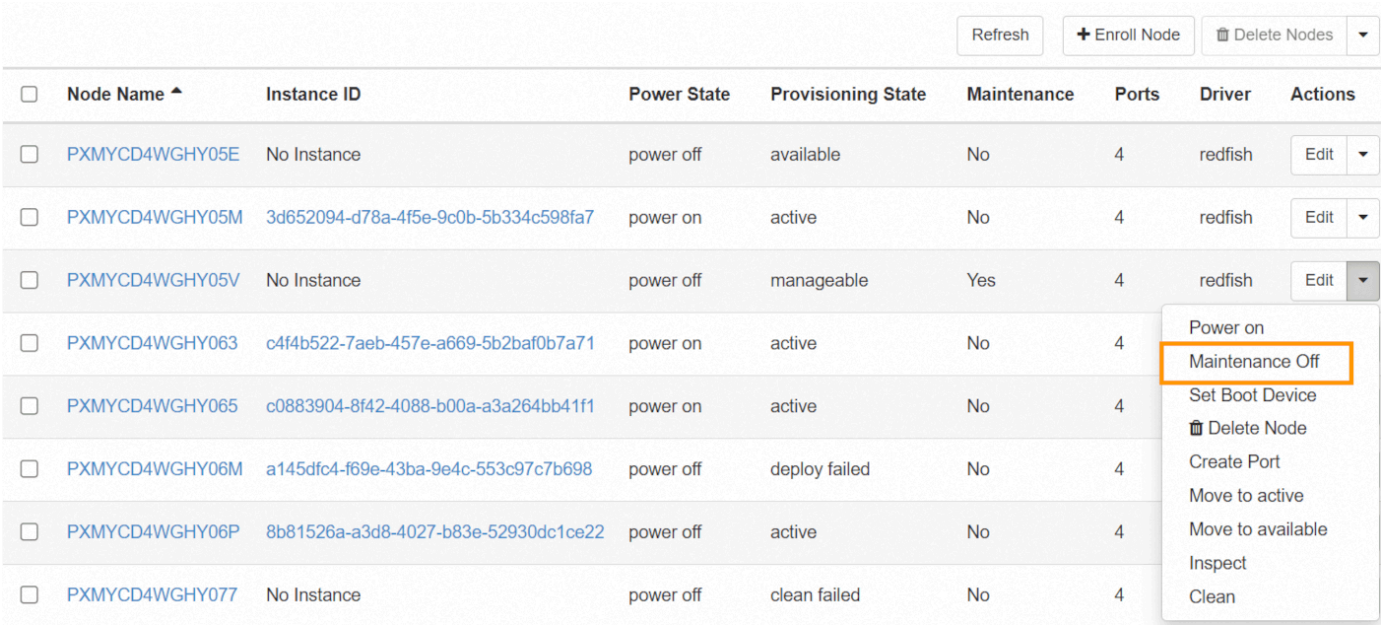
From the Horizon interface:



When enabling maintenance, you may specify a reason so that the team responsible for the nodes has the information. This reason is optional.



Once your maintenance operations are complete, you can disable maintenance:



From the OpenStack APIs:

```
baremetal node maintenance set $BAREMETAL_NODE_ID --reason "Maintenance reason"
```

You can then retrieve the maintenance status and the reason using the following command:

```
baremetal node show $BAREMETAL_NODE_ID
```

You will find the following lines:

```
| maintenance             | True
| maintenance_reason      | "Maintenance reason"
```

To remove the node from maintenance, use the command:

```
baremetal node maintenance unset $BAREMETAL_NODE_ID
```

References

- OpenStack Official Documentation - Horizon
- OpenStack Ironic States
- OpenStack Ironic Troubleshooting - Maintenance

Go further

If you need training or technical assistance for the implementation of our solutions, contact your sales representative or click this link to request a quote and have your project analyzed by our Professional Services team experts.

Join our community of users.

How to use the APIs and obtain the credentials

Discover the steps required to configure Keycloak and the OpenStack CLI to allow authentication via Keycloak

Objective

OPCP integrates a centralized authentication with **Keycloak**. It is therefore necessary to configure the **OpenStack CLI** so that it uses Keycloak as the identity provider (Identity Provider).

This guide describes the steps required to configure Keycloak and the OpenStack CLI to allow authentication via Keycloak.

Requirements

- Be an administrator of the OPCP infrastructure and have access to the administration interface (admin.dashboard).
- Have access to the Keyloack admin interface.
- Have a user with sufficient rights to log in to Horizon on the OPCP offer.

Instructions

Creating a Keycloak client for the OpenStack CLI

A **dedicated Keycloak client** is required to allow the OpenStack CLI to securely communicate with the Keycloak server.

Steps

1. **Log in to the Keycloak administration interface**

- Log in to your Keycloak instance and select the *realm* in which the OpenStack users are defined.

2. **Create a new client**

- Go to the `Clients` section and click on **Create a client**.
- Enter a **Client ID**, for example:

```
openstack-cli
```

- Click on **Next**.

3. **Enable client authentication**

- Enable **Client Authentication** (set to **ON**).
- Click on **Next**, then on **Save**.

4. **Configure scopes (Client Scopes)**

- Open the `Client Scopes` tab.

- Select the scope named:

```
[your-client-id]-dedicated
```

- Click on **Configure a new mapper**.

5. **Add a user group attribute mapper**

- Choose the mapper type **aggregated-user-group-attribute-mapper**.
- Configure the following fields:

Field	Value
Name	`projects`
User Attribute	`project`
Token Claim Name	`projects`

- Click on **Save**.

6. **Add an audience mapper** (from OPCP version 3.0.5 onwards)

> **Info**
> As of OPCP version **3.0.5**, you must add an audience mapper to the client so that the issued tokens include the `keystone` audience. Without this mapper, authentication against Keystone fails.

- From the `Client Scopes` tab, select the `[your-client-id]-dedicated` scope again.
- Click on **Configure a new mapper** and choose the **Audience** type.
- Configure the following fields:

Field	Value
Name	`keystone-audience`
Included Client Audience	`keystone`
Included Custom Audience	*(leave empty)*

- Click on **Save**.

7. **Retrieve the client credentials**

- Go to the `Credentials` tab of the client you just created.
- Copy and securely store the **Client Secret** — it will be needed when configuring the OpenStack CLI.

Configuration of the OpenStack CLI

Once the Keycloak client is created, the OpenStack CLI must be configured to use this client as the OIDC (OpenID Connect) identity provider.

Steps

1. **Install the OpenStack CLI tools**

If not already done:

```
sudo pip install python-openstackclient
```

2. **Set environment variables for Keycloak authentication**

Example:

```
export OS_INTERFACE=public
export OS_IDENTITY_API_VERSION=3
export OS_AUTH_URL="https://keystone.domain.ovh"
export OS_AUTH_TYPE="v3oidcpassword"
export OS_PROTOCOL="openid"
export OS_IDENTITY_PROVIDER="keycloak-admin"
export OS_CLIENT_ID="keycloak-client-id"
export OS_CLIENT_SECRET="keycloak-client-credentials"
export OS_DISCOVERY_ENDPOINT="https://admin.keycloak.domain.ovh/realms/master/.well-known/openid-configuration"
export OS_USERNAME="keycloak-user-username"
export OS_PASSWORD="keycloak-user-password"
export OS_PROJECT_ID="project-id"
```

> **Tip**
>
> You can use the following script to easily generate the openrc.sh configuration file:

```
 #!/usr/bin/env bash

read -p "Your environment's base FQDN (e.g. example.bmp.ovhgoldorack.ovh): " FQDN_ENV

read -p 'master or pod realm ? (master/pod): ' REALM
if [ "$REALM" != "master" ] && [ "$REALM" != "pod" ]; then
    echo "Invalid input. Please enter either 'master' or 'pod'."
    exit 1
fi

read -p 'Keycloak client ID: ' KC_CLIENT_ID
read -srp 'Keycloak client secret: ' KC_CLIENT_SECRET && echo

read -p 'Keycloak username: ' KC_USERNAME_INPUT
read -srp 'Keycloak password: ' KC_PASSWORD_INPUT && echo

read -p 'Openstack Project ID (not the name): ' PROJECT_ID

printf "\n\nHere is your configuration, paste it to your shell or use the generate openrc.sh file\n\n"
cat << EOM
export OS_INTERFACE=public
export OS_IDENTITY_API_VERSION=3
export OS_AUTH_URL="https://keystone.${FQDN_ENV}"
export OS_AUTH_TYPE="v3oidcpassword"
export OS_PROTOCOL="openid"
export OS_IDENTITY_PROVIDER=$([ "$REALM" = "master" ] && echo "keycloak-admin" || echo "keycloak")
export OS_CLIENT_ID="$KC_CLIENT_ID"
export OS_CLIENT_SECRET="$KC_CLIENT_SECRET"
export OS_DISCOVERY_ENDPOINT="https://$([ "$REALM" = "master" ] && echo "admin.keycloak" || echo
"keycloak").${FQDN_ENV}\`/realms\`/$REALM/.well-known/openid-configuration"
export OS_USERNAME="$KC_USERNAME_INPUT"
export OS_PASSWORD="$KC_PASSWORD_INPUT"
export OS_PROJECT_ID="$PROJECT_ID"
EOM

echo "#!/usr/bin/env bash

export OS_INTERFACE=public
export OS_IDENTITY_API_VERSION=3
export OS_AUTH_URL="https://keystone.${FQDN_ENV}"
export OS_AUTH_TYPE="v3oidcpassword"
export OS_PROTOCOL="openid"
export OS_IDENTITY_PROVIDER=$([ "$REALM" = "master" ] && echo "keycloak-admin" || echo "keycloak")
export OS_CLIENT_ID="$KC_CLIENT_ID"
export OS_CLIENT_SECRET="$KC_CLIENT_SECRET"
export OS_DISCOVERY_ENDPOINT="https://$([ "$REALM" = "master" ] && echo "admin.keycloak" || echo
"keycloak").${FQDN_ENV}\`/realms\`/$REALM/.well-known/openid-configuration"
export OS_USERNAME="$KC_USERNAME_INPUT"
export OS_PASSWORD="$KC_PASSWORD_INPUT"
export OS_PROJECT_ID="$PROJECT_ID > $PROJECT_ID."-openrc.sh"
```

Tip

Proxy configuration:

If you are using a proxy to access your service, you must configure your environment variables to take this proxy into account.

To do this, add the following commands lines:

```
export https_proxy=http://your-adress-ip:port/
export http_proxy=http://your-adress-ip:port/
```

Configuration verification

You can test your configuration using a few simple commands:

```
openstack token issue
openstack project list
openstack server list
```

If these commands return results, the **Keycloak ↔ OpenStack** integration is correctly configured.

Troubleshooting

Problem	Possible cause	Solution
`Invalid client credentials`	Wrong or missing `Client Secret`	Check the secret in the **Credentials** tab of the Keycloak client
`Unauthorized`	The user is not associated with the correct group or project	Check the `project` attributes of the user in Keycloak
`OIDC discovery failed`	Wrong URL in `DISCOVERY_ENDPOINT`	Make sure it points to the correct Keycloak *realm*

References

- Keycloak Documentation – OpenID Connect
- OpenStack Keystone Documentation

Go further

If you need training or technical assistance for the implementation of our solutions, contact your sales representative or click this link to request a quote and have your project analyzed by our Professional Services team experts.

Join our community of users.

How to install an instance from the Horizon interface

Learn how to install an OPCP instance via Horizon by configuring networks, subnets, instances, and SSH keys.

Objective

Before deploying services on your **OPCP** arrays, you must have an installed and active instance.

This guide details the steps to install an OPCP instance from the **Horizon** interface.

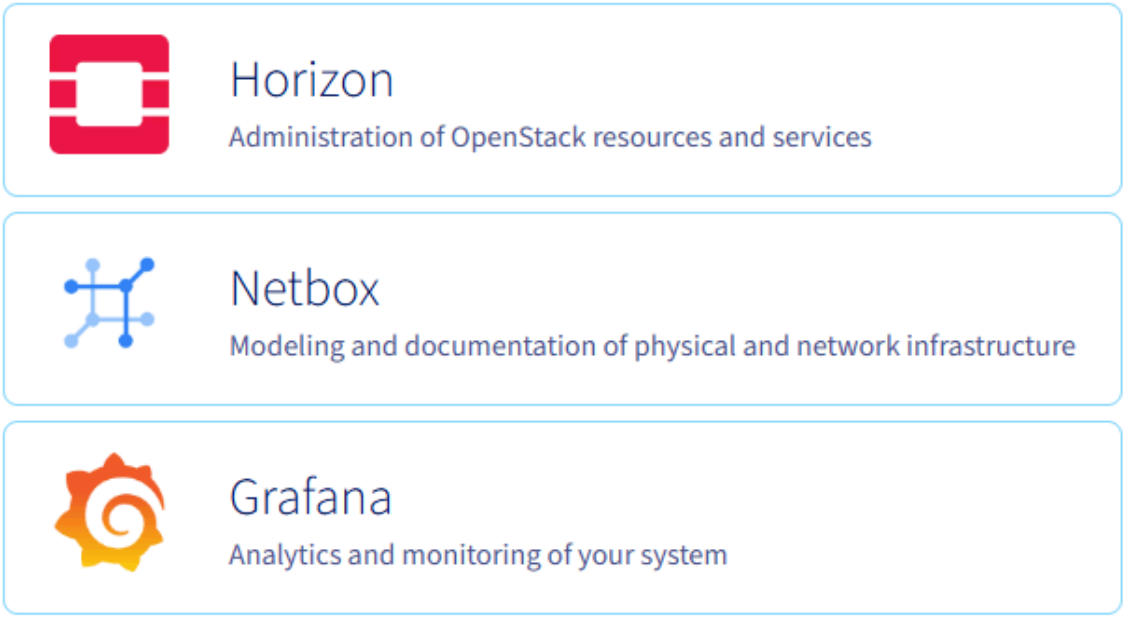
Requirements

- Have an active OPCP service.
- Have a **user account** with sufficient permissions to access Horizon for the OPCP offer.

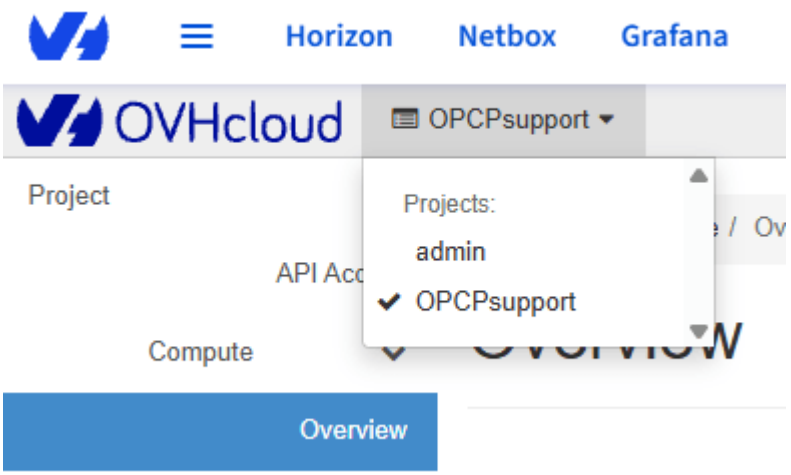
Instructions

1. Log in to Horizon

Log in to the **Horizon** interface of your OPCP environment.



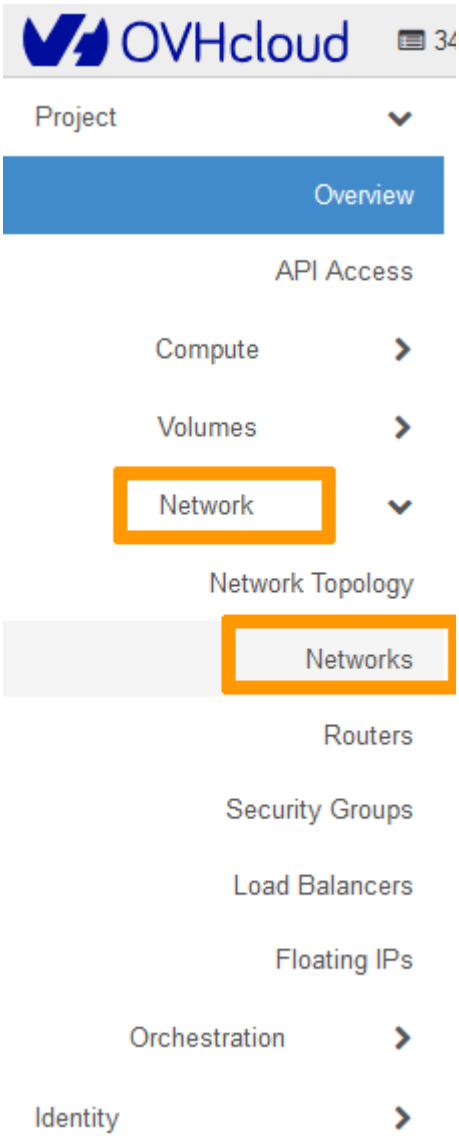
Once connected, select the **project** where you want to install your instance.



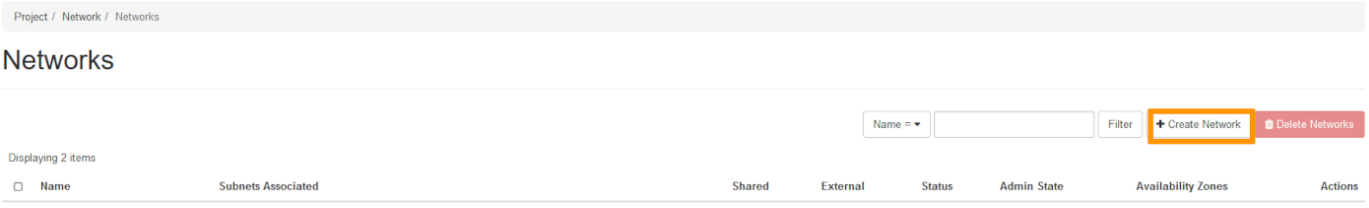
2. Creating a private network

Before deploying your instance, it is generally necessary to create a **private network** so that it can be accessed within your local infrastructure.

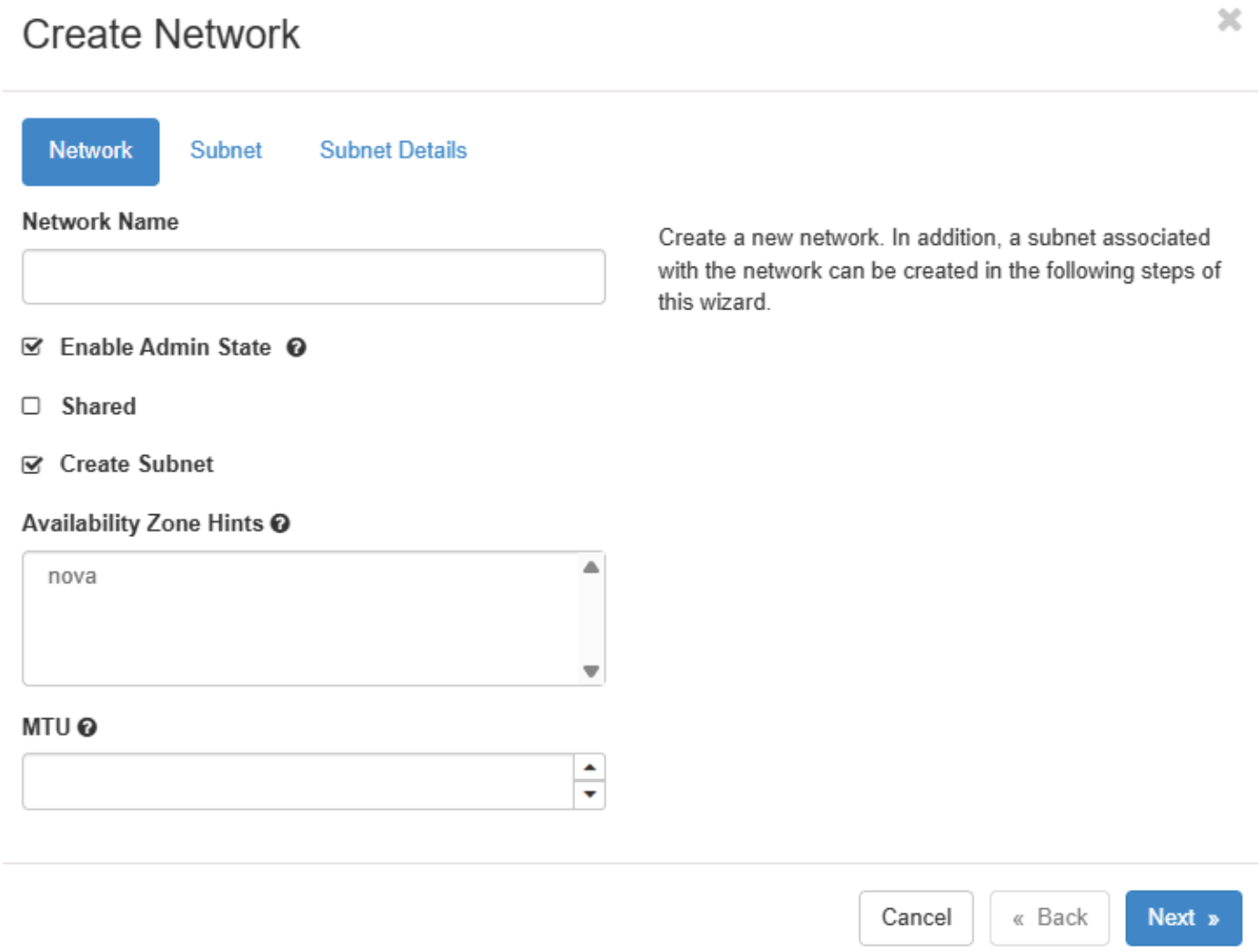
In the left-hand menu, click on **Network**, then on **Networks**.



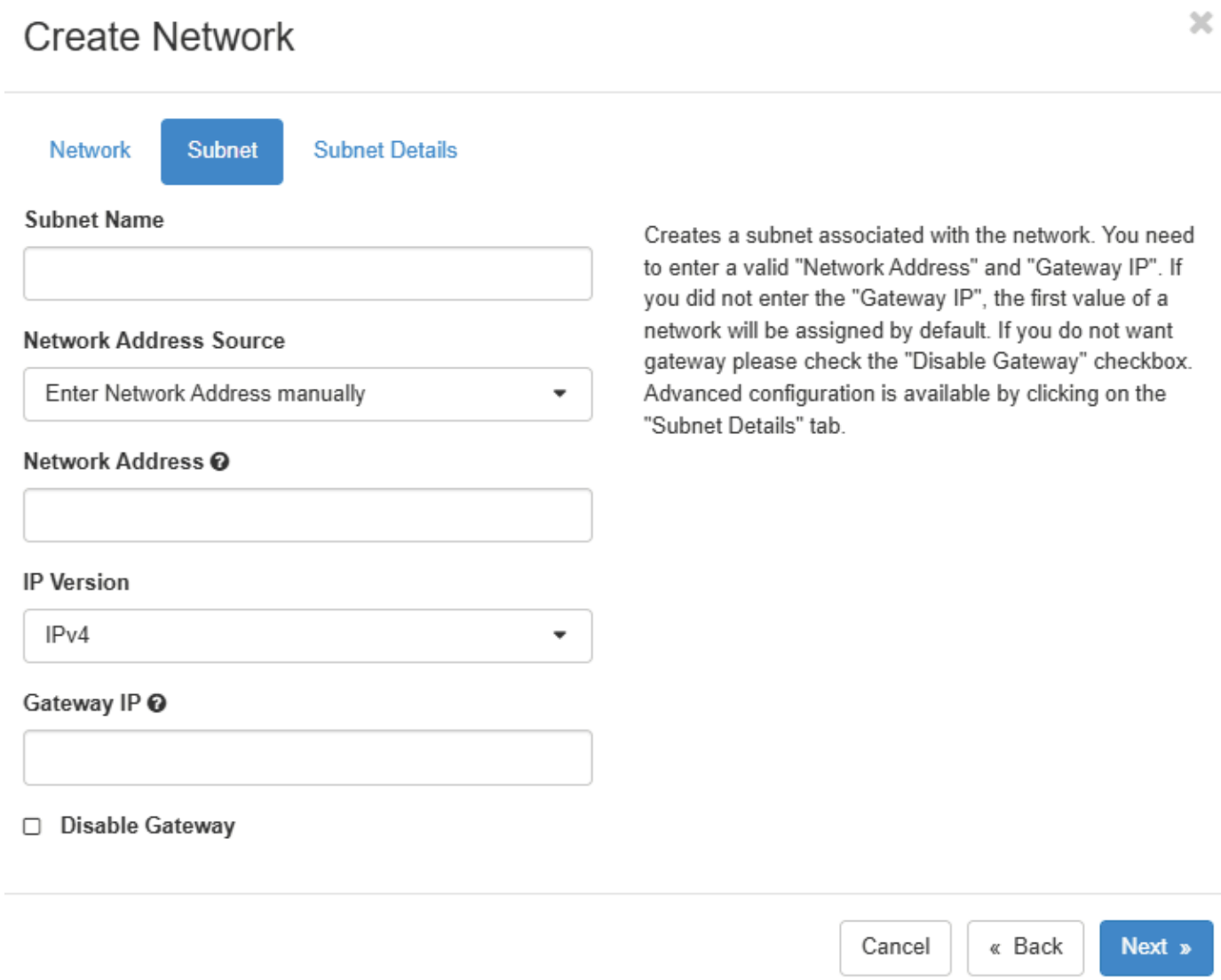
Click on **Create Network**.



Network
Subnet
Subnet Details



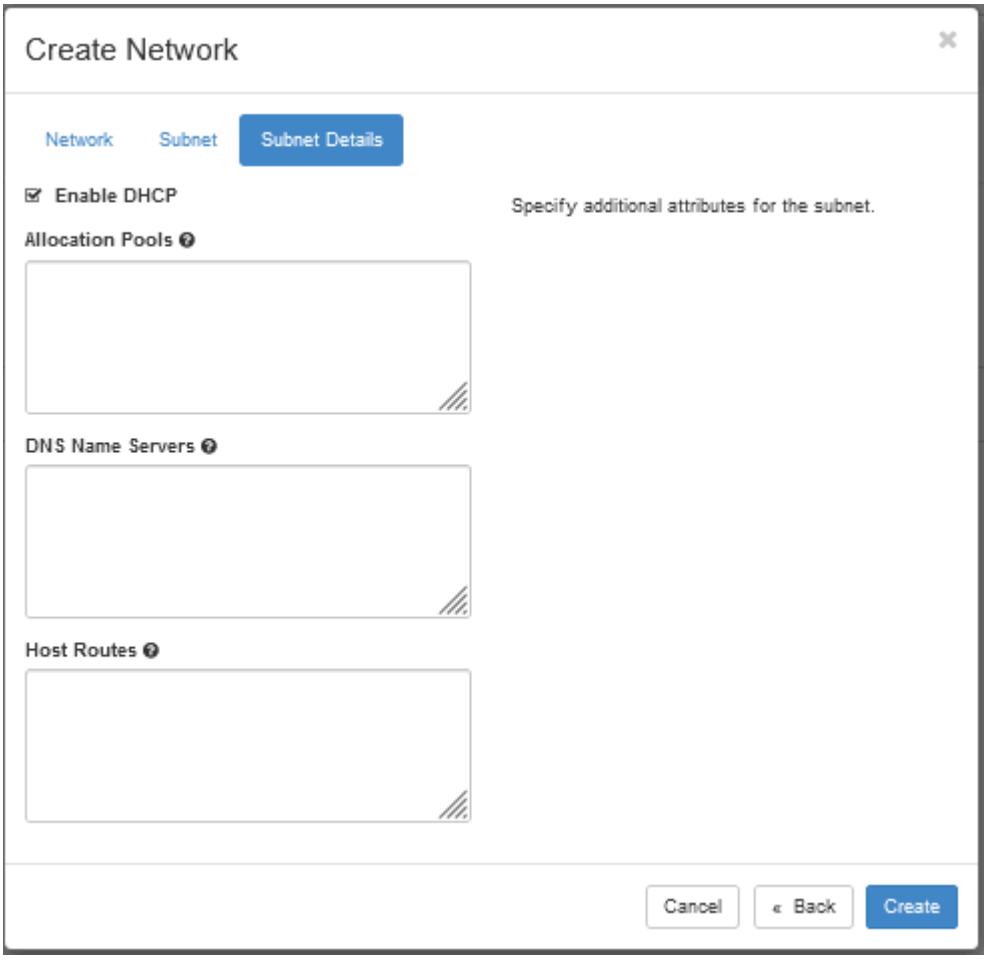
Field	Description
Network Name	Enter a name for your network.
Enable Admin State	Leave this option checked to activate the network.
Shared	Check this box if you want to make the network available to multiple projects.
Create Subnet	Leave this option checked to create a subnet.
Availability Zone Hints	Leave the default value.



Info

Although it is possible to create a network without a subnet, it cannot be attached to an instance unless it has one.

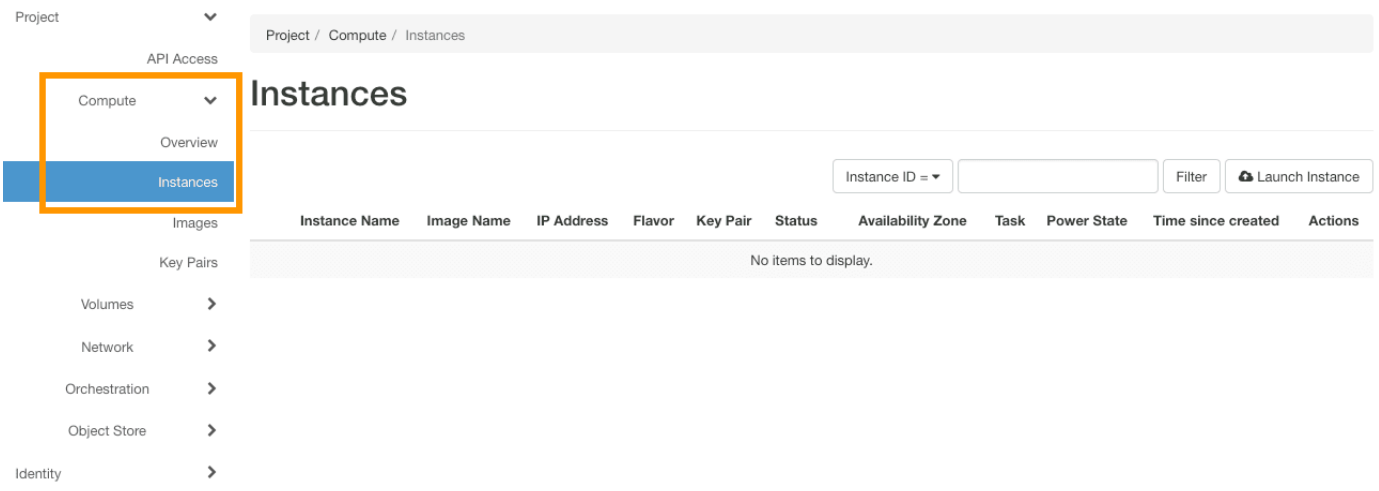
Field	Description
Subnet Name	Enter a name for your subnet.
Network Address	Define a private IP range, for example `192.168.100.0/24`.
IP Version	Leave the default value **IPv4**.
Gateway IP	Optional. If not specified, an address will be automatically selected.
Disable Gateway	Check this box to not assign a gateway address.



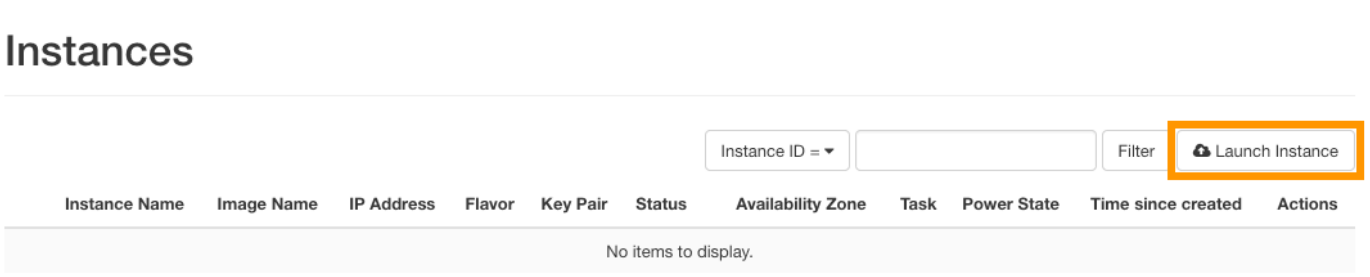
Field	Description
Enable DHCP	Keep enabled if you want IP addresses to be automatically assigned.
Allocation Pools	Optional. Allows you to define a specific IP address range.
DNS Name Servers	Optional. Lets you specify one or more DNS servers.
Host Routes	Optional. Allows you to add static routes.

3. Creating an instance

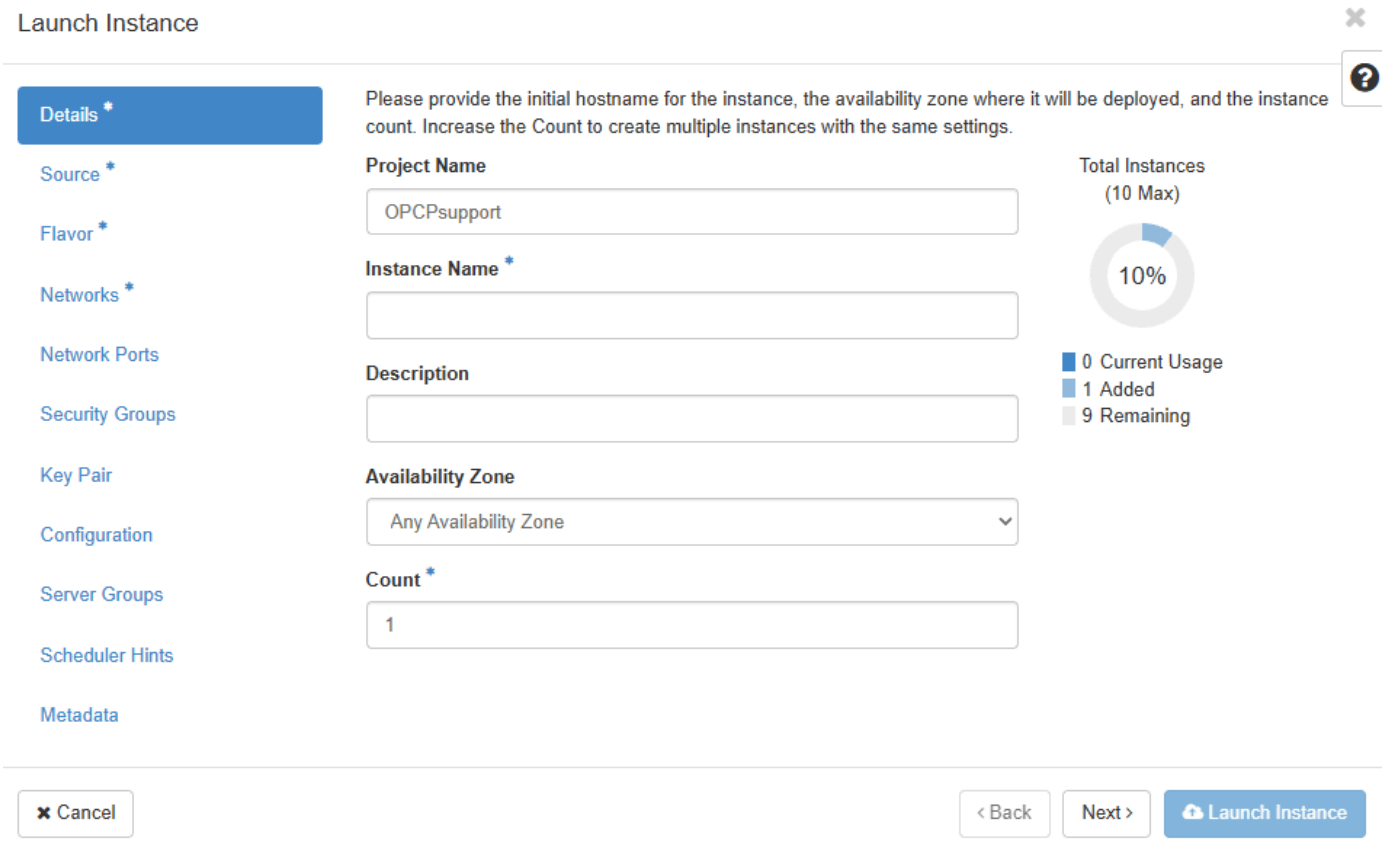
In the left-hand menu, click on **Compute**, then on **Instances**.



Click on **Launch Instance** to create a new instance.

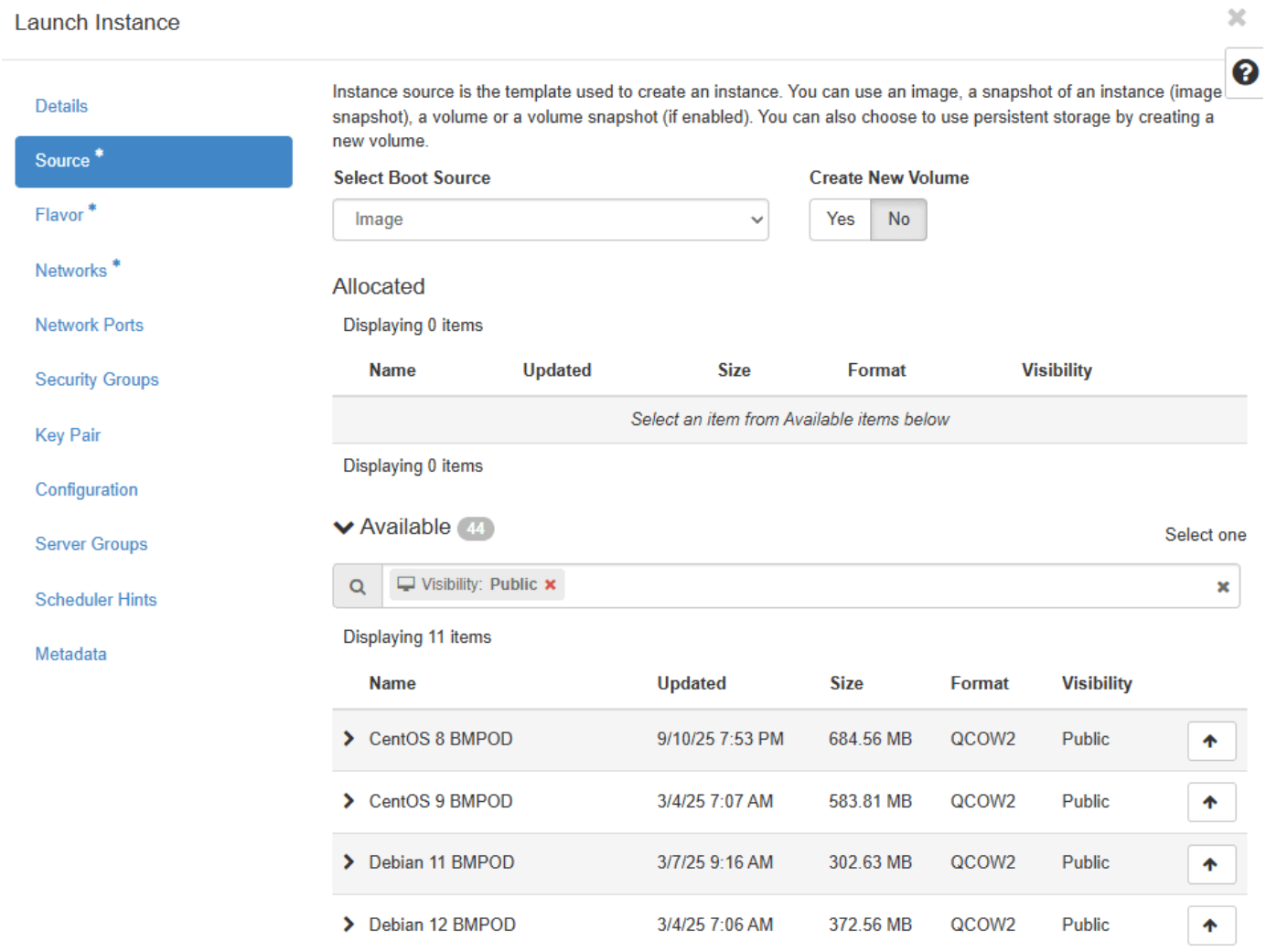


Details



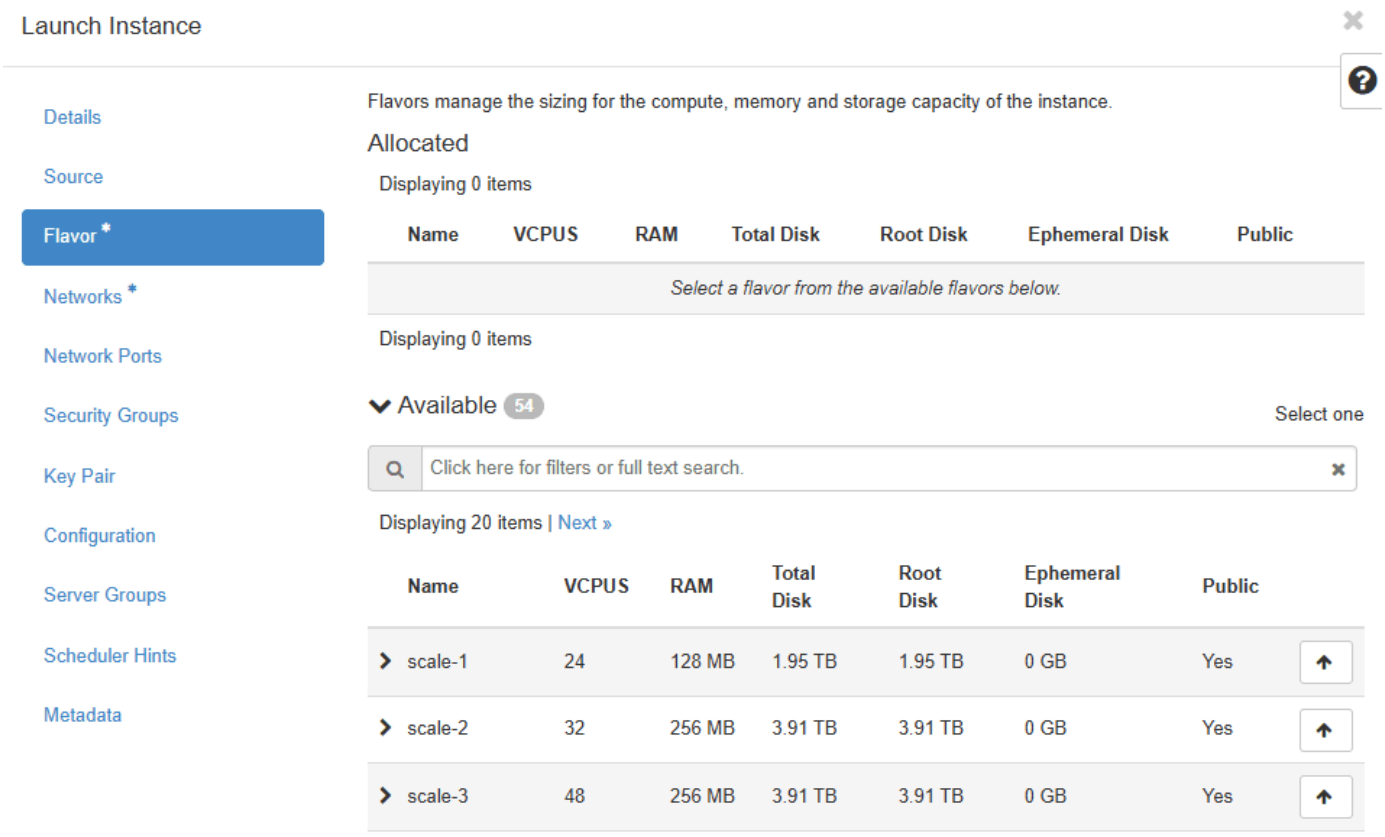
Field	Description
Instance Name	Enter the name of the instance to create.
Description	Optional. Add a description if needed.
Availability Zone	Leave the default value **nova**.
Count	Specify the number of instances to deploy.

Source



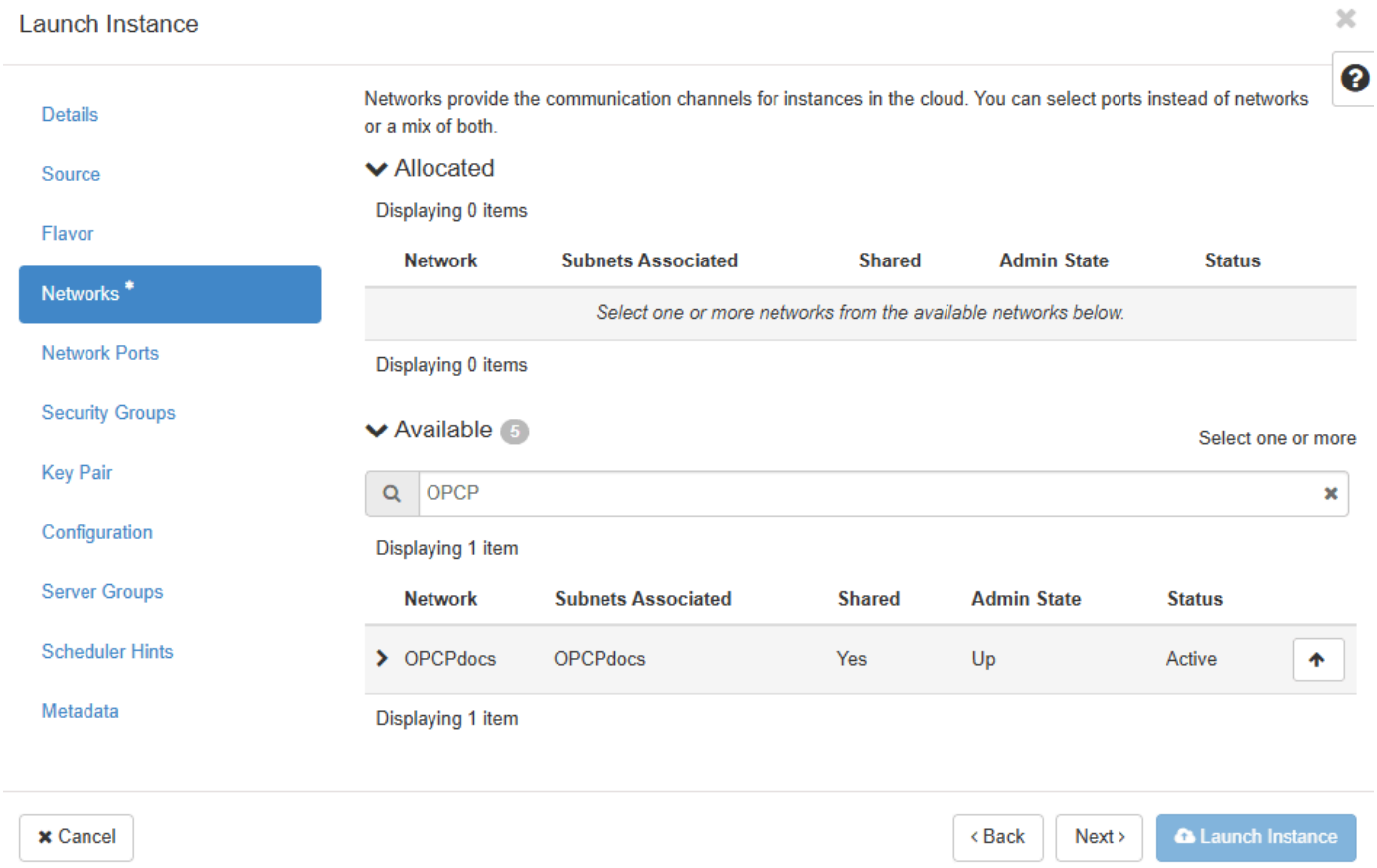
Field	Description
Boot Source	Select the boot source: *Image* or *Instance Snapshot*.
Image Name	Choose the image to use (e.g. *Debian 12 BMPOD*).

Flavor



Select the appropriate **hardware configuration** (vCPU, memory, storage).

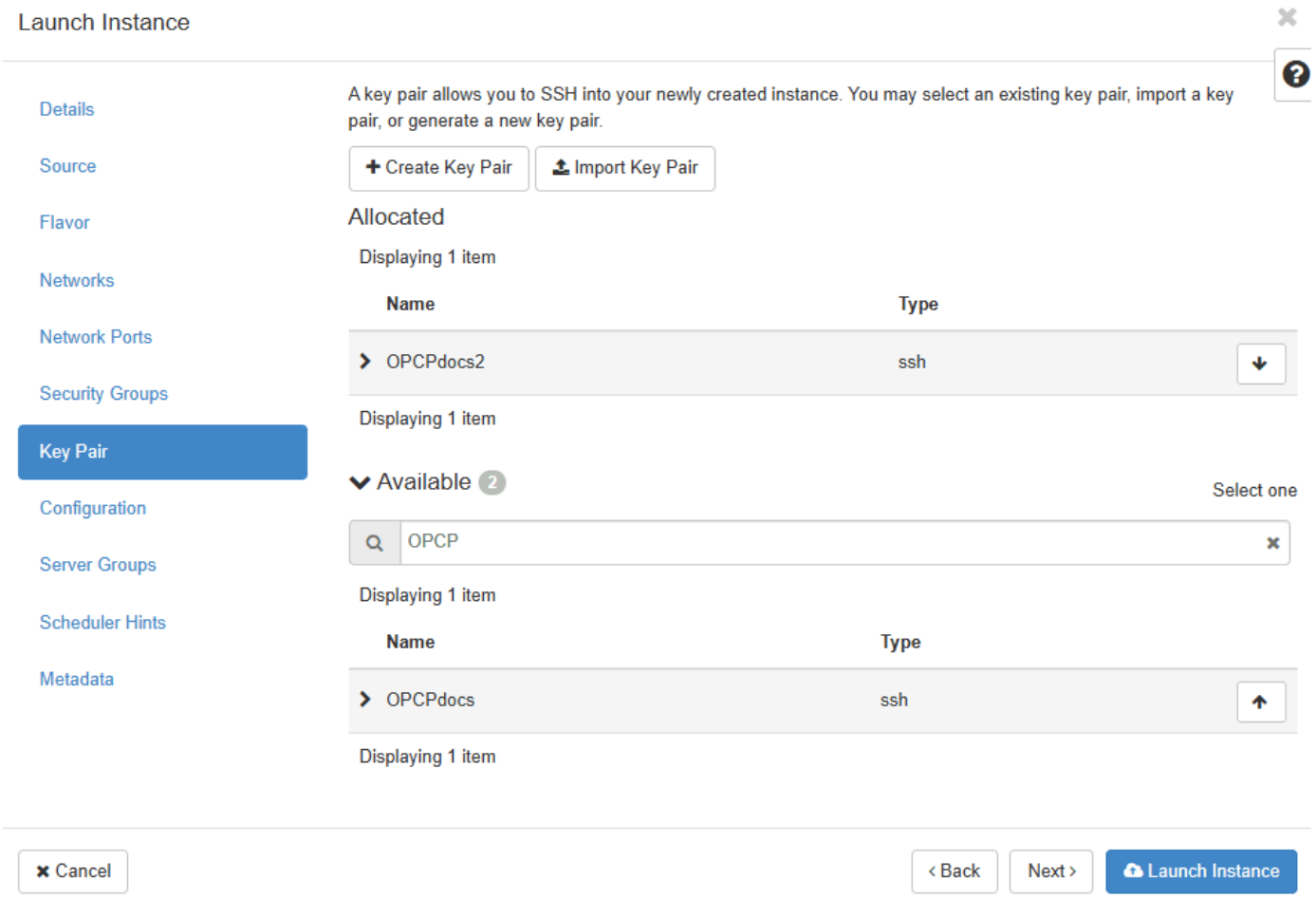
Networks



Select the **private network** previously created. You can also attach an existing **network port** from the *Network Ports* tab.

4. Managing SSH key pairs

> **Warning**
> Although selecting an SSH key in Horizon is not mandatory, it is **essential for connecting to the instance** once it has been created.

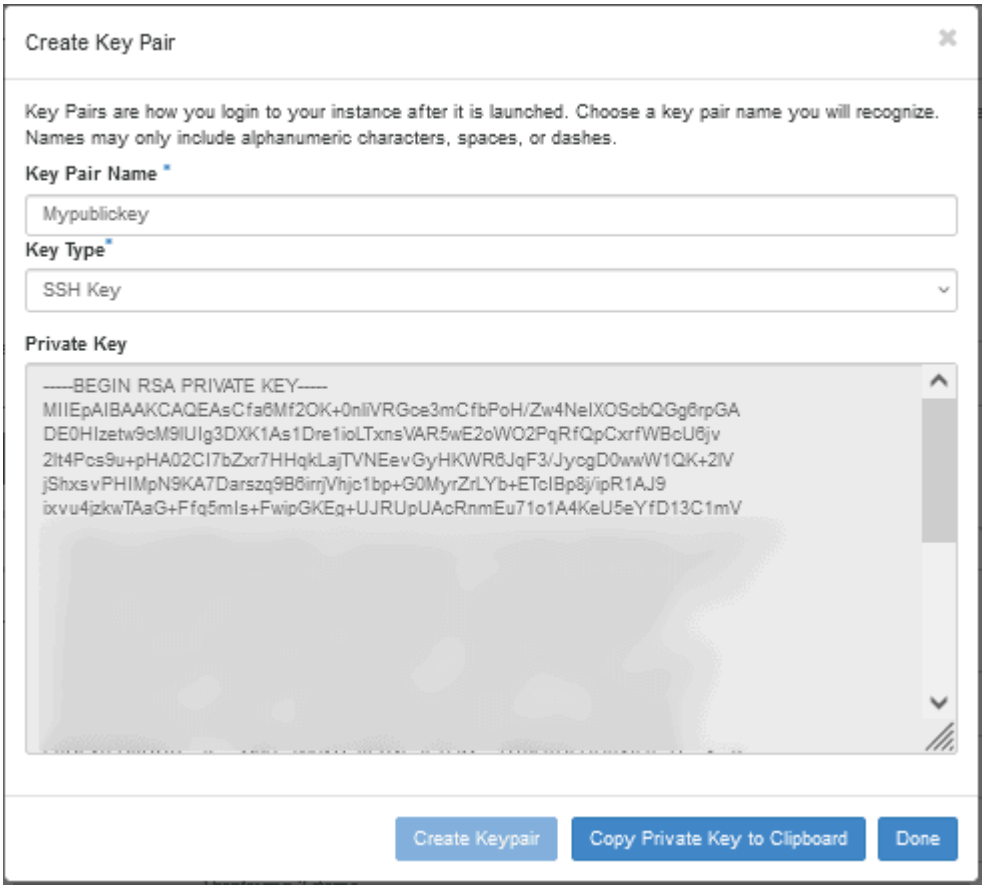


Create a new key pair
Import an existing key

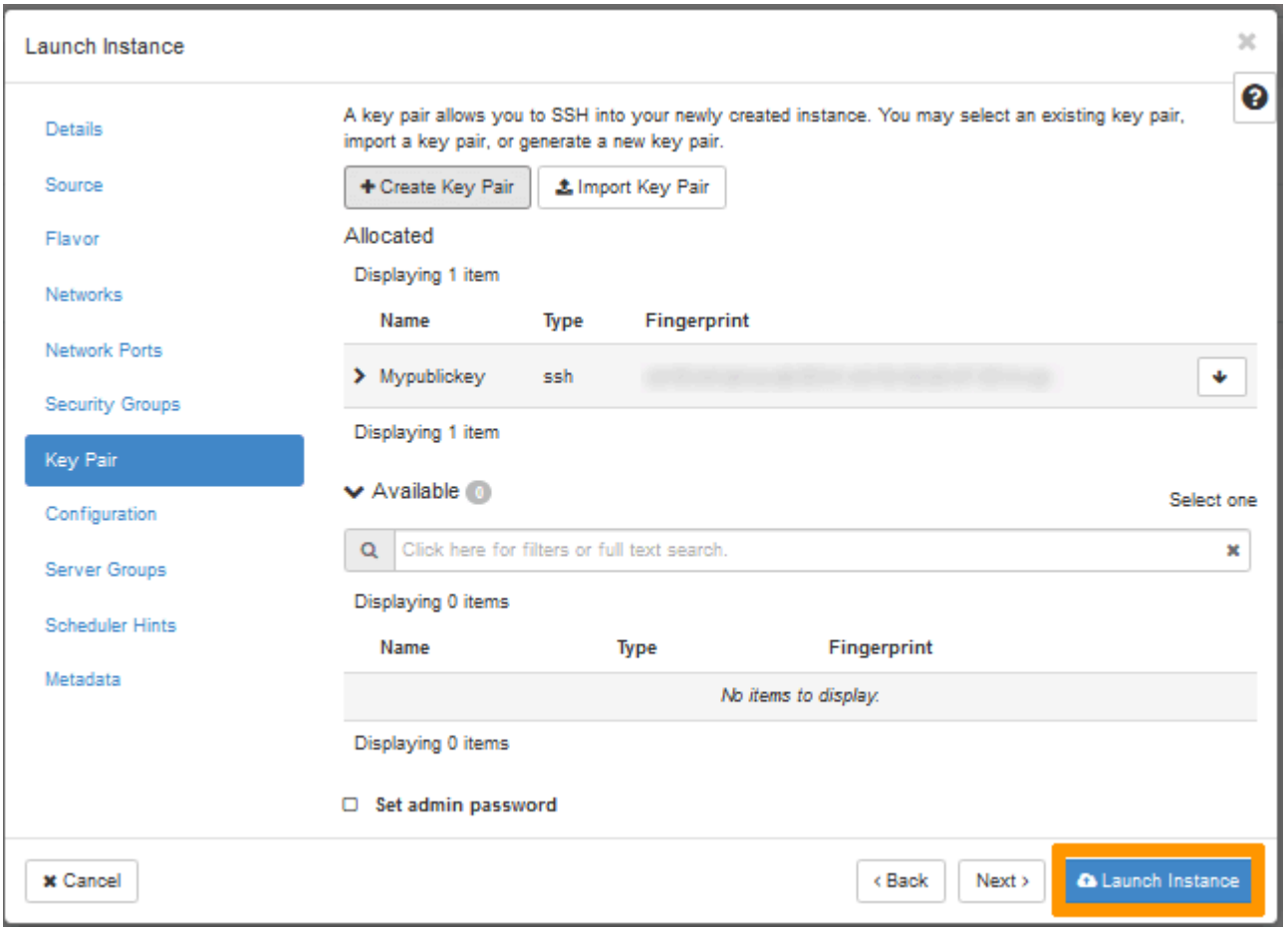
Click **+ Create Key Pair** and fill in the following fields:

Field	Description
Key Pair Name	Enter a name for the key.
Key Type	Select **SSH Key**.

Next, click on **Create Keypair**. Copy the private key with **Copy Private Key to Clipboard**, then click on **Done**.



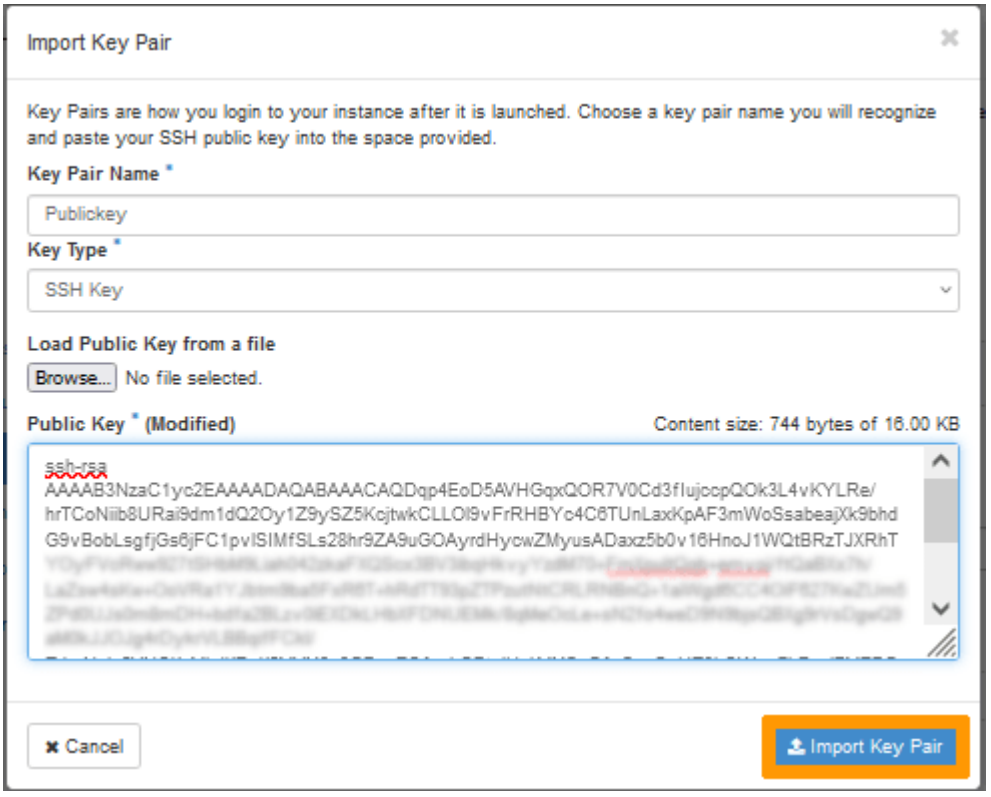
The key is now selected by default. Click on **Launch Instance** to start creating the instance.



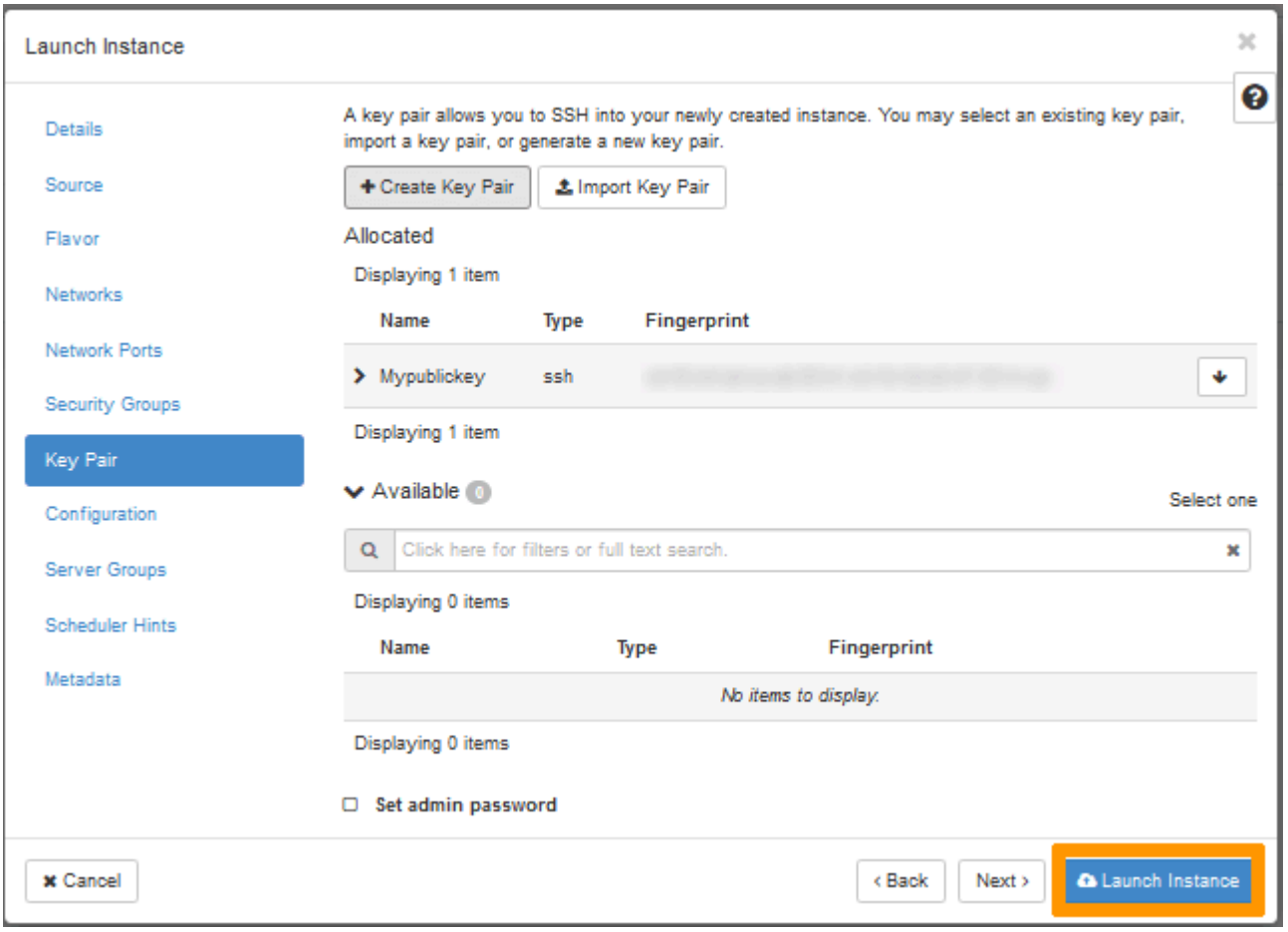
Click on **Import Key Pair** and fill in the following fields:

Field	Description
Key Pair Name	Name of the key.
Key Type	Select **SSH Key**.
Public Key	Paste your public key or upload the corresponding file.

Click on **Import Key Pair**.



The key is now selected by default. Click on **Launch Instance** to start creating the instance.



5. Other options

The other configuration tabs (Security Groups, Configuration, Metadata, etc.) are not required for a standard installation.
To learn more, refer to the official OpenStack documentation.

6. References

- OpenStack Official Documentation – Horizon
- OpenStack Networking Guide (Neutron)
- OpenStack Compute Guide (Nova)
- OpenStack Key Pairs
- Debian 12 Official Site

Go further

If you need training or technical assistance for the implementation of our solutions, contact your sales representative or click this link to request a quote and have your project analyzed by our Professional Services team experts.

Join our community of users.

How to deploy an instance via the OpenStack CLI

Find out how to deploy an OPCP instance via the OpenStack command-line interface by configuring networks, subnets, instances, and SSH keys

Objective

This guide details the steps to install an OPCP node by creating an instance through the OpenStack CLI. Before deploying services on your OPCP clusters, you must have at least one installed and active node.

Requirements

- Have an active OPCP service.
- Have a user account with sufficient rights to access the OpenStack APIs.
- Prepare the environment to use the OpenStack API.
- Load the environment variables for the project.

Instructions

You can obtain a list of available OpenStack commands using the following command:

```
openstack command list
```

You can filter the displayed commands by specifying a group:

```
openstack command list --group compute
```

You can also get information about a specific command by adding `help` before it:

```
openstack help flavor list

usage: openstack flavor list [-h] [-f {csv,json,table,value,yaml}] [-c COLUMN]
                             [--quote {all,minimal,none,nonnumeric}] [--noindent]
                             [--max-width `<integer>`] [--fit-width] [--print-empty]
                             [--sort-column SORT_COLUMN]
                             [--sort-ascending | --sort-descending] [--public | --private | --all]
                             [--min-disk `<min-disk>`] [--min-ram `<min-ram>`] [--long]
                             [--marker `<flavor-id>`] [--limit `<num-flavors>`]

List flavors ...
```

> **Tip**
>
> Check the OpenStack client documentation directly on the OpenStack website.

Retrieve the parameters needed to create an instance

Create a private network and a subnet

Step 1: Create the Private Network

Before deploying your instance, it is generally necessary to create a **private network** so that it is accessible within your local infrastructure. If you already have a network with a subnet in your project that you want to use, you can skip this creation step and directly list your networks to get the name or ID of the desired network.

```
openstack network create $NETWORK_NAME
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   |                                      |
| availability_zones        |                                      |
| created_at                | 2025-11-05T15:18:06Z                 |
| description               |                                      |
| dns_domain                | None                                 |
| id                        | da5ed9ae-65c2-441b-99f4-75d8eb87c214 |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| is_default                | False                                |
| is_vlan_transparent       | None                                 |
| mtu                       | 1500                                 |
| name                      | opcp-docs                            |
| port_security_enabled     | True                                 |
| project_id                | 057ac6e82ade49078a5c66f4371eaf22     |
| provider:network_type     | vlan                                 |
| provider:physical_network | physnet1                             |
| provider:segmentation_id  | 2140                                 |
| qos_policy_id             | None                                 |
| revision_number           | 1                                    |
| router:external           | Internal                             |
| segments                  | None                                 |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tags                      |                                      |
| updated_at                | 2025-11-05T15:18:06Z                 |
+---------------------------+--------------------------------------+
```

By default, a network is visible only to the project that created it (as well as to administrator users). If you want to create a network **shared across all your projects**, you can use the `--share` parameter. To share a network only with specific projects, you must use OpenStack's **Role-Based Access Control (RBAC)** mechanism: Neutron RBAC Documentation.

Additionally, if you want to create the network in a **specific VLAN**, you can specify it with the following parameters:

- `--provider-network-type vlan`
- `--provider-physical-network physnet1`
- `--provider-segment $VLAN_ID`

For example, if you want to create a shared private network in VLAN 2025 named `opcpdocs`:

```
openstack network create --share --provider-network-type vlan --provider-physical-network physnet1 --provider-segment
2025 opcpdocs
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   |                                      |
| availability_zones        |                                      |
| created_at                | 2025-11-05T15:34:30Z                 |
| description               |                                      |
| dns_domain                | None                                 |
| id                        | 2a92f464-e1c0-4daa-8ecd-b3ba6b3b96da |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| is_default                | False                                |
| is_vlan_transparent       | None                                 |
| mtu                       | 1500                                 |
| name                      | opcpdocs                             |
| port_security_enabled     | True                                 |
| project_id                | 07f0c728fe844d778cda63ca28547937     |
| provider:network_type     | vlan                                 |
| provider:physical_network | physnet1                             |
| provider:segmentation_id  | 2025                                 |
| qos_policy_id             | None                                 |
| revision_number           | 1                                    |
| router:external           | Internal                             |
| segments                  | None                                 |
| shared                    | True                                 |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tags                      |                                      |
| updated_at                | 2025-11-05T15:34:30Z                 |
+---------------------------+--------------------------------------+
```

Once the network is created, you can list it using the command:

```
openstack network list --name $NETWORK_NAME
+--------------------------------------+-----------+---------+
| ID                                   | Name      | Subnets |
+--------------------------------------+-----------+---------+
| 2a92f464-e1c0-4daa-8ecd-b3ba6b3b96da | opcp-docs |         |
+--------------------------------------+-----------+---------+
```

If needed, you can list all networks by removing the `--name` argument.

Step 2: Create the Subnet

By default, the only required information to create a subnet on your network is the CIDR you want to configure and the network you just created:

```
openstack subnet create --network $NETWORK_NAME --subnet-range 192.168.120.0/24 $SUBNET_NAME
```

However, if you want to specify an allocation pool, you can define it using various parameters.
For example, to create a subnet with CIDR 192.168.120.0/24, allocating only 50 IP addresses from the CIDR, and using a specific gateway, you can use the following command:

```
openstack subnet create --network opcpdocs --subnet-range 192.168.120.0/24 --allocation-pool
start=192.168.120.11,end=192.168.120.60 --gateway 192.168.120.8 opcpdocs-subnet
+----------------------+--------------------------------------+
| Field                | Value                                |
+----------------------+--------------------------------------+
| allocation_pools     | 192.168.120.11-192.168.120.60        |
| cidr                 | 192.168.120.0/24                     |
| created_at           | 2025-11-05T16:17:58Z                 |
| description          |                                      |
| dns_nameservers      |                                      |
| dns_publish_fixed_ip | None                                 |
| enable_dhcp          | True                                 |
| gateway_ip           | 192.168.120.8                        |
| host_routes          |                                      |
| id                   | b532e25f-3aae-4396-99e9-ad6811a03a6e |
| ip_version           | 4                                    |
| ipv6_address_mode    | None                                 |
| ipv6_ra_mode         | None                                 |
| name                 | opcpdocs-subnet                      |
| network_id           | 2a92f464-e1c0-4daa-8ecd-b3ba6b3b96da |
| project_id           | 07f0c728fe844d778cda63ca28547937     |
| revision_number      | 0                                    |
| segment_id           | None                                 |
| service_types        |                                      |
| subnetpool_id        | None                                 |
| tags                 |                                      |
| updated_at           | 2025-11-05T16:17:58Z                 |
+----------------------+--------------------------------------+
```

This subnet can be used to deploy an instance, allowing OpenStack to assign an IP address to it during installation.

Adding a public SSH key

First, you need to add a public SSH key to connect to your instances.

- List the SSH key-related commands:

```
openstack help | grep keypair
  keypair create  Create new public or private key for server ssh access
  keypair delete  Delete public or private key(s)
  keypair list    List key fingerprints
  keypair show    Display key details
```

- Add the public SSH key:

```
openstack keypair create --public-key ~/.ssh/id_rsa.pub $SSHKEY
```

- List available SSH keys:

```
openstack keypair list
+---------------+-------------------------------------------------+------+
| Name          | Fingerprint                                     | Type |
+---------------+-------------------------------------------------+------+
| SSHKEY        | 5c:fd:9d:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:3a | ssh  |
+---------------+-------------------------------------------------+------+
```

List instance flavors

Next, retrieve the ID or name of the flavor you want to use:

```
openstack flavor list
+--------------------------------------+--------------------+---------+------+-----------+-------+-----------+
| ID                                   | Name               |     RAM | Disk | Ephemeral | VCPUs | Is Public |
+--------------------------------------+--------------------+---------+------+-----------+-------+-----------+
| 4e731d33-4c1e-4286-bc63-68c2cba3bc59 | gpu                |     400 |  900 |         0 |   128 | True      |
| 8bc6b1e0-dfdd-44d8-83fb-cef5e8fb6ec2 | scale-2            |     256 | 4000 |         0 |    32 | True      |
| f975e49a-fd75-4a7e-868e-2842972e82e2 | hsm                |     512 | 4000 |         0 |   128 | True      |
| fcd2c556-9a16-4532-a5c8-e9d9275ba78f | scale-1            |     128 | 2000 |         0 |    24 | True      |
| ff5b4d7d-46c9-4bf5-b300-f1f465661e64 | scale-3            |     256 | 4000 |         0 |    48 | True      |
+--------------------------------------+--------------------+---------+------+-----------+-------+-----------+
```

List available images

Finally, get the ID or name of the image to use for the instance:

```
openstack image list
+--------------------------------------+-----------------------------------------------+--------+
| ID                                   | Name                                          | Status |
+--------------------------------------+-----------------------------------------------+--------+
| 6540686f-0150-496b-a894-03d95ef8bc7e | ironic-agent.initramfs                        | active |
| 5f6d4237-023b-4a25-8ceb-7051ffc6d632 | ironic-agent.kernel                           | active |
| 7600f9fc-fc1f-4fb8-aaa3-878bb9399d26 | CentOS 9 OPCP                                 | active |
| ebabdaaa-a52d-4246-9790-e77ba5c49519 | Debian 12 LVM OPCP                            | active |
+--------------------------------------+-----------------------------------------------+--------+
```

Instance installation

With the previously gathered information, you can create an instance to deploy an operating system along with your SSH public key on the desired flavor:

```
openstack server create --key-name OPCPdocs2 --flavor scale-1 --image "Debian 12 LVM OPCP" --network opcpdocs
OPCPdocs-server
+-------------------------------------+----------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
---------------------------------------------+
| Field                               | Value
|
+-------------------------------------+----------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------
---------------------------------------------+
| OS-DCF:diskConfig                   | MANUAL
|
| OS-EXT-AZ:availability_zone         | None
|
| OS-EXT-SRV-ATTR:host                | None
|
| OS-EXT-SRV-ATTR:hostname            | opcpdocs-server
|
| OS-EXT-SRV-ATTR:hypervisor_hostname | None
|
| OS-EXT-SRV-ATTR:instance_name       | None
|
| OS-EXT-SRV-ATTR:kernel_id           | None
|
| OS-EXT-SRV-ATTR:launch_index        | None
|
| OS-EXT-SRV-ATTR:ramdisk_id          | None
|
| OS-EXT-SRV-ATTR:reservation_id      | r-900xkw3k
|
| OS-EXT-SRV-ATTR:root_device_name    | None
|
| OS-EXT-SRV-ATTR:user_data           | None
|
| OS-EXT-STS:power_state              | N/A
|
| OS-EXT-STS:task_state               | scheduling
|
| OS-EXT-STS:vm_state                 | building
|
| OS-SRV-USG:launched_at              | None
|
| OS-SRV-USG:terminated_at            | None
|
| accessIPv4                          | None
|
| accessIPv6                          | None
|
| addresses                           | N/A
|
| adminPass                           | VbRFzvGnk9rP
|
| config_drive                        | None
|
| created                             | 2025-11-05T16:41:32Z
|
| description                         | None
|
| flavor                              | description=, disk='2000', ephemeral='0',
extra_specs.:architecture='bare_metal', extra_specs.resources:CUSTOM_DISCOVERED='1',
extra_specs.resources:DISK_GB='0', extra_specs.resources:MEMORY_MB='0', extra_specs.resources:VCPU='0',          |
|                                     | extra_specs.trait:CUSTOM_SCALE1='required', id='scale-1', is_disabled=,
is_public='True', location=, name='scale-1', original_name='scale-1', ram='128', rxtx_factor=, swap='0', vcpus='24'
|
```

```
| hostId                              | None
|
| host_status                         | None
|
| id                                  | 2f9c2e41-b4dc-4787-a3ef-9b2b2d686dbb
|
| image                               | Debian 12 LVM OPCP (ebabdaaa-a52d-4246-9790-e77ba5c49519)
|
| key_name                            | OPCPdocs2
|
| locked                              | None
|
| locked_reason                       | None
|
| name                                | OPCPdocs-server
|
| pinned_availability_zone            | None
|
| progress                            | None
|
| project_id                          | 07f0c728fe844d778cda63ca28547937
|
| properties                          | None
|
| security_groups                     | name='default'
|
| server_groups                       | None
|
| status                              | BUILD
|
| tags                                |
|
| trusted_image_certificates          | None
|
| updated                             | 2025-11-05T16:41:32Z
|
| user_id                             | b1481b3f72e8aecda9a623b215085227f9d85170f4fc4524b3a4ab1a0b97dfde
|
| volumes_attached                    |
|
+-------------------------------------+--------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------
-------------------------------------------+
```

By default, the instance to be installed is automatically selected from the pool of nodes in `Available` status, for which the **required traits** of the flavor match the requested installation. This means a physical server meeting the constraints described by the **traits** will be selected.

After a few minutes, the instance is deployed, and you can list your installed instances using the following command:

```
openstack server list
+--------------------------------------+-----------------+--------+------------------------+---------------------+--
-------+
| ID                                   | Name            | Status | Networks               | Image               |
Flavor  |
+--------------------------------------+-----------------+--------+------------------------+---------------------+--
-------+
| 2f9c2e41-b4dc-4787-a3ef-9b2b2d686dbb | OPCPdocs-server | ACTIVE | opcpdocs=192.168.120.59 | Debian 12 LVM OPCP  |
scale-1 |
+--------------------------------------+-----------------+--------+------------------------+---------------------+--
-------+
```

If you want to install the instance on a specific node, you can specify the node ID in your command:

```
openstack server create --flavor $flavor_ID --image $image_ID --network $network_ID --key-name $your_keyname --
availability-zone nova::$baremetal_node_ID $server_name
```

You must ensure that the node is `Available` and has the required **traits** to deploy the desired flavor.

To check the current state of a node and retrieve its ID, see the OPCP Node Lifecycle guide.

Rebooting or stopping an instance

You can manage your instance's power state directly through the OpenStack API:

```
# Soft reboot
openstack server reboot $INSTANCE_ID

# Hard reboot
openstack server reboot --hard $INSTANCE_ID

# Stop the instance
openstack server stop $INSTANCE_ID

# Start the instance
openstack server start $INSTANCE_ID
```

> **Warning**
>
> Stopping an instance must always be done through the OpenStack API or the Horizon interface, not from within the operating system using commands such as `systemctl poweroff`. An action performed directly from the OS is not propagated to OpenStack, which would continue to consider the instance as active, leaving it in an inconsistent state between its actual status and the one known by the platform. If a node remains stuck in an intermediate state after a shutdown performed from the OS, you can unblock it by forcing it off and then on again with the commands `openstack baremetal node power off <node>` and `openstack baremetal node power on <node>`.

Deleting an instance

You can delete an instance with the following command:

```
openstack server delete $INSTANCE_ID
```

Your node will go into `Cleaning` state. This step involves a hardware reset of the physical server and erasure of the data on its disks. During this step, the instance will no longer appear in the instance list; however, the node will not be immediately available for a new installation. Please consider this delay during your maintenance operations. The operation may take several minutes before the node is `Available` again and ready for a new deployment.

References

- OpenStack Official Documentation - Client
- OpenStack Official Documentation - Network

Go further

If you need training or technical assistance for the implementation of our solutions, contact your sales representative or click this link to request a quote and have your project analyzed by our Professional Services team experts.

Join our community of users.

How to setup LACP on a Node

Learn how to setup a node in OpenStack to use LACP (Link Aggregation Control Protocol)

Objective

LACP configuration must be applied to the node before deploying an instance so that the network interfaces are properly aggregated.

This guide explains how to configure a node (physical server) in OPCP to enable LACP (Link Aggregation Control Protocol).

We will also see how to configure **bonding** (logical grouping of multiple network interfaces into a single virtual interface) within your instance to fully leverage **LACP**.

> **Warning**
>
> Standard users cannot setup LACP by themselves.
> You must be an **admin**, or have **available nodes** in your OpenStack project.
>
> It is recommended to setup LACP **before** deploying an instance.
> This guide **does not cover** configuring a node that is already in production.

Why Use LACP?

LACP can be used in two specific use cases:

- **Increase network capacity:** By aggregating multiple network cards, you can add the bandwidth of each NIC to the aggregate, for the same networks.
- **Increase resilience:** Each OPCP server has 4×25 G interfaces and is connected to two network switches (A & B) to ensure resilience in case of hardware failure. LACP allows you to create virtual interfaces that remain available in the event of hardware failure by aggregating two NICs connected to different switches.

Requirements

Before starting, make sure you have the following:

- An active OPCP service.
- **Configured OpenStack CLI access** with the necessary permissions (`clouds.yaml` or environment variables).
- The **admin** role and/or transferred nodes within your project.

LACP is an advanced networking configuration requiring system and networking expertise. Apply this guide only if you are already familiar with: configuring nodes in OpenStack Ironic, managing ports in OpenStack Neutron, and using the OpenStack CLI.

Instructions

Node Configuration

1. List Nodes

The list of available nodes in your project can be displayed with the following command:

```
openstack baremetal node list
```

Example output:

```
+--------------------------------------+---------------+--------------------------------------+-------------+-------
-------------+-------------+
| UUID                                 | Name          | Instance UUID                        | Power State |
Provisioning State | Maintenance |
+--------------------------------------+---------------+--------------------------------------+-------------+-------
-------------+-------------+
| 88830859-5b16-4935-8f41-d381b754cbe5 | SERVER-1      | None                                 | power off   |
available          | False       |
| 726bb7e9-3b20-4a44-99d7-2af747983781 | SERVER-2      | None                                 | power off   |
available          | False       |
| af711edf-a579-491e-b474-3c03639ed99b | SERVER-3      | 83b7083b-bbdd-4327-941c-ee91197818c5 | None        | active
| True        |
+--------------------------------------+---------------+--------------------------------------+-------------+-------
-------------+-------------+
```

2. Transfer Node Ownership (Admin Only)

An admin can transfer node ownership to a specific project:

```
openstack baremetal node set <node-id> --owner <project-id>
```

3. List Network Ports

Each physical network interface card (NIC) of a node is represented in OpenStack as a port.

To list the ports associated with a node:

```
openstack baremetal port list --node <node-id>
```

Example output:

```
+--------------------------------------+-------------------+
| UUID                                 | Address           |
+--------------------------------------+-------------------+
| 068a06b2-ebf9-48c9-a3c3-94016ca5e3da | 85:32:f2:87:29:da |
| 4937d704-7517-4525-86d1-abecb94a7ce9 | 85:32:f2:87:29:db |
| 422eece6-dcfa-40cd-975d-ba8bb11c774e | 85:32:f2:89:66:f8 |
| 34073903-92ad-47d1-a751-15aa96991415 | 85:32:f2:89:66:f9 |
+--------------------------------------+-------------------+
```

To view detailed information about a port, including physical connection info:

This is particularly useful if you plan to configure a **2x2 LACP bonding**, distributing NICs across two switches for **better redundancy and fault tolerance**.

```
openstack baremetal port show <port-id>
```

Example output:

```
openstack baremetal port show 71899d54-546d-4fdd-8d8b-52ad986bf425
+----------------------+--------------------------------------------------------------------------------------------
----+
| Field                | Value
|
+----------------------+--------------------------------------------------------------------------------------------
----+
| address              | 85:32:f2:89:66:f9
|
| created_at           | 2025-01-06T10:43:38.020574+00:00
|
| extra                | {}
|
| internal_info        | {}
|
| is_smartnic          | False
|
| local_link_connection | {'switch_id': 'c0:00:15:8c:33:78', 'port_id': 'Ethernet25/2', 'switch_info': 'demo-tor1b-
a70'} |
| node_uuid            | ddc7c763-74fc-4900-b9e6-5463233836b0
|
| physical_network     | None
|
| portgroup_uuid       | c3d3fcb3-7a92-4257-b944-736d222e8c4a
|
| pxe_enabled          | False
|
| updated_at           | 2025-11-06T09:52:46.355883+00:00
|
| uuid                 | 71899d54-546d-4fdd-8d8b-52ad986bf425
|
+----------------------+--------------------------------------------------------------------------------------------
----+
```

4. Enable Maintenance Mode

Before any network configuration change, the node must be placed in **maintenance mode** to prevent deployment during the operation:

```
openstack baremetal node maintenance set <node-id>
```

5. Create a Port Group (LACP Bond)

A **port group** allows enabling LACP aggregation between multiple network interfaces.

> **Tip**
>
> You can create:
>
> - a **single port group** for a 1×4 bond, or
> - two **port groups** for 2×2 bonds.

Use the `--mode 802.3ad` parameter to enable LACP.

The `--address <MAC>` parameter must be equal to the MAC address of the PXE port only if it's being used. Otherwise, you can omit the parameter or set the MAC value from one of the physical interfaces you will use.

You can list all ports with `openstack baremetal port list --node <node> --long` and verify if PXE is used or not.

Note: We recommend prefixing the portgroup name with the node name for clearer identification when listing portgroups (e.g. `<node-name>-<name>`).

Example:

```
openstack baremetal port group create \
  --node 88830859-5b16-4935-8f41-d381b754cbe5 \
  --name node_name-pg-lacp \
  --mode 802.3ad \
  --address 00:00:00:20:00:01
```

Example output:

```
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| uuid                      | d082c2ab-5960-44e3-920d-3d6dfb6811e9 |
| address                   | 00:00:00:20:00:01                    |
| node_uuid                 | 88830859-5b16-4935-8f41-d381b754cbe5 |
| name                      | node_name-pg-lacp                    |
| mode                      | 802.3ad                              |
| standalone_ports_supported | True                                 |
+---------------------------+--------------------------------------+
```

6. Associate Ports to the Group

Each port on the node must be linked to the created port group:

```
openstack baremetal port set --port-group `<port-group-id>` `<port-id>`
```

Example:

```
openstack baremetal port set --port-group d082c2ab-5960-44e3-920d-3d6dfb6811e9 068a06b2-ebf9-48c9-a3c3-94016ca5e3da
openstack baremetal port set --port-group d082c2ab-5960-44e3-920d-3d6dfb6811e9 4937d704-7517-4525-86d1-abecb94a7ce9
openstack baremetal port set --port-group d082c2ab-5960-44e3-920d-3d6dfb6811e9 422eece6-dcfa-40cd-975d-ba8bb11c774e
openstack baremetal port set --port-group d082c2ab-5960-44e3-920d-3d6dfb6811e9 34073903-92ad-47d1-a751-15aa96991415
```

7. Disable Maintenance Mode

Once configuration is complete, disable maintenance mode:

```
openstack baremetal node maintenance unset `<node-id>`
```

8. Create an Instance on the configured Node

After configuring your node with LACP, you can deploy an instance.

By default, OpenStack selects a host based on the **flavor** and **scheduler rules**, which doesn't guarantee it will use the node you just configured.

To ensure your instance is deployed on that specific node, target it using its **availability zone**:

```
openstack server create --availability-zone "nova::`<node-id>`"
```

Example:

```
openstack server create --image `<image-name>` \
   --nic net-id=NETWORK1,v4-fixed-ip=198.18.56.200 \
   --flavor `<flavor-id>` \
   --key-name `<keypair-name>`  \
   --availability-zone "nova::`<node-id>`" \
   `<instance-name>`
```

Summary of Steps

Step	Action	Command
1	List nodes	`openstack baremetal node list`
2	Transfer ownership	`openstack baremetal node set <node-id> --owner <tenant-id>`
3	List ports	`openstack baremetal port list --node <node-id>`
4	Enable maintenance mode	`openstack baremetal node maintenance set <node-id>`
5	Create LACP group	`openstack baremetal port group create --node <node-id> --name <port-group-name> --mode 802.3ad`
6	Associate ports	`openstack baremetal port set --port-group <port-group-id> <port-id>`
7	Disable maintenance mode	`openstack baremetal node maintenance unset <node-id>`
8	Create instance	`openstack server create --image <image-name> --nic net-id=<network-1> --flavor <flavor-id> --key-name <keypair-name> --availability-zone "nova::<node-id>" <instance-name>`

Instance Operating System Configuration

After configuring your node in OpenStack and deploying an OS, you need to configure networking to leverage **bonding**, which aggregates multiple NICs for higher performance and redundancy.

Check Bonding Configuration

On some images (like **Debian 12** or **Ubuntu 22.04**), **bonding** is automatically detected and configured. However, other distributions may require manual adjustments.

1. Check Active Bonds

```
ls /proc/net/bonding/
bond0
```

2. Check Member Interfaces

```
cat /proc/net/bonding/bond0 | grep Interface
Slave Interface: ens22f1np1
Slave Interface: ens22f0np0
Slave Interface: ens21f1np1
Slave Interface: ens21f0np0
```

3. Check Transmit Hash Policy

```
cat /proc/net/bonding/* | grep Trans
Transmit Hash Policy: layer2 (0)
```

> **Warning**
>
> To fully utilize available bandwidth, configure the policy to `layer3+4` (some OSes default to `layer2`, which is less efficient). For more information: Ubuntu Bonding Documentation.

Modify Configuration

1. Temporary Change (Non-Persistent)

To test your configuration manually:

```
sudo ip link set bond0 type bond xmit_hash_policy layer3+4
```

Note: This configuration will reset on reboot.

2. Persistent Change (Example with Netplan and cloud-init)

Create the configuration file (e.g. `/etc/cloud/cloud.cfg.d/99-custom-network.cfg`) and include:

```
network:
  version: 2
  ethernets:
    ens21f0np0: {}
    ens21f1np1: {}
    ens22f0np0: {}
    ens22f1np1: {}
  bonds:
    bond0:
      dhcp4: true
      interfaces:
        - ens21f0np0
        - ens21f1np1
        - ens22f0np0
        - ens22f1np1
      macaddress: 00:00:00:20:00:23
      parameters:
        mode: 802.3ad
        transmit-hash-policy: layer3+4
        lacp-rate: fast
        mii-monitor-interval: 100
```

Then apply the configuration by rebooting the instance.

3. Test Bandwidth with `iperf3`

To properly test LACP, you need **two nodes** in the **same network**, both configured with LACP.

Iperf3 Server (Node 1)

```
iperf3 -s
```

Iperf3 Client (Node 2)

Use `-P` to generate multiple parallel streams to reach maximum throughput.

```
iperf3 -c <node-ip> -P 64
```

Example Result:

```
[SUM] 0.0000-10.0121 sec   110 GBytes  94.0 Gbits/sec
```

With a 4×25 Gbps link, you should achieve around **100 Gbps**. Some system tuning might be required to fully reach this capacity.

Conclusion

You have successfully configured:

- **LACP (802.3ad)** at the OpenStack Baremetal node level;
- **Bonding configuration** within the guest OS;
- And validated **network performance** using `iperf3`.

Your instance is now ready to leverage the full available bandwidth of the aggregated link.

Go further

If you need training or technical assistance for the implementation of our solutions, contact your sales representative or click this link to request a quote and have your project analyzed by our Professional Services team experts.

Join our community of users.

How to set up Trunk ports on a Node

Learn how to configure Neutron Trunk ports in OPCP for multi-network vlan connectivity on bare metal or virtual machine instances

Objective

Trunk ports allow a single instance (bare metal or virtual machine) to send and receive traffic on multiple Neutron networks using vlan tagging, through a single physical interface or an LACP bond.

This guide explains how to configure Neutron Trunk ports in OPCP to enable multi-network (vlan) connectivity on a bare metal node or a virtual machine.

This guide also shows how to configure **vlan sub-interfaces** within your instance to access each network attached to the trunk.

> **Warning**
>
> Trunk creation requires the **admin** role. A project user cannot create trunks.
> Adding sub-ports to a trunk also requires admin rights by default, but this can be delegated by your administrator.
>
> It is recommended to configure the trunk **before** deploying an instance.
> This guide **does not cover** configuring a trunk on an instance that is already in production.

Why Use Trunk Ports?

Trunk ports can be used in three specific use cases:

- **Multi-network access from a single instance:** Trunk ports allow a bare metal server or a virtual machine to communicate on multiple isolated Neutron networks using vlan tagging, without needing separate ports for each network.
- **Overcome physical interface limits on bare metal:** On a bare metal server, the number of Neutron networks is normally limited by the number of physical network interfaces. With trunk ports, you can connect to more networks than available physical interfaces by multiplexing multiple vlans over a single interface or LACP bond.
- **Simplified network management:** Instead of provisioning multiple ports and attaching them individually, you create a single trunk with sub-ports, each tagged with a specific vlan ID. This keeps the network topology clean and manageable.

Requirements

Before starting, ensure you have the following:

- An active OPCP service.
- **Configured OpenStack CLI access** with the necessary permissions (`clouds.yaml` or environment variables).

- The **admin** role (required for trunk creation and sub-port management).
- At least **two Neutron networks** already created in your project (one for the parent port and one or more for sub-ports).
- An available bare metal node or virtual machine project.

Trunk port configuration is an advanced networking feature requiring familiarity with OpenStack Neutron networking concepts, vlan tagging, and the OpenStack CLI.

Instructions

Network and Trunk Configuration

1. Identify Your Networks

Before creating the trunk, identify the networks your instance needs access to. List the available networks in your project:

```
openstack network list
```

Example output:

```
+--------------------------------------+--------------------+--------------------------------------+
| ID                                   | Name               | Subnets                              |
+--------------------------------------+--------------------+--------------------------------------+
| 3fa85f64-5717-4562-b3fc-2c963f66afa6 | primary-network    | a1b2c3d4-e5f6-7890-abcd-ef1234567890 |
| 7c9e6679-7425-40de-944b-e07fc1f90ae7 | network-1          | b2c3d4e5-f6a7-8901-bcde-f12345678901 |
| 9b1deb4d-3b7d-4bad-9bdd-2b0d7b3dcb6d | network-2          | c3d4e5f6-a7b8-9012-cdef-123456789012 |
+--------------------------------------+--------------------+--------------------------------------+
```

2. Create the Parent Port

Create a Neutron port that will serve as the **parent port** of the trunk. This port is required by the Neutron trunk model to anchor the trunk to the instance.

```
openstack port create --network <network-name> <parent-port-name>
```

Example:

```
openstack port create --network primary-network primary-port
```

Example output:

```
+-------------------------+--------------------------------------+
| Field                   | Value                                |
+-------------------------+--------------------------------------+
| id                      | f47ac10b-58cc-4372-a567-0e02b2c3d479 |
| mac_address             | fa:16:3e:aa:bb:cc                    |
| name                    | primary-port                         |
| network_id              | 3fa85f64-5717-4562-b3fc-2c963f66afa6 |
| status                  | DOWN                                 |
+-------------------------+--------------------------------------+
```

> **Warning**
>
> On **bare metal instances**, the parent port is a **dummy port**. It exists in the Neutron database but has **no effect on the network fabric**. The network assigned to the parent port will **not** carry any traffic to the instance. All actual network connectivity must be configured through **sub-ports** (see steps 4 and 5).
>
> On **virtual machines**, the parent port carries the parent network as **untagged** traffic on the base interface. Sub-port networks are delivered as tagged vlan traffic.

3. Create the Trunk

Create a Neutron trunk using the parent port created in the previous step:

```
openstack network trunk create --parent-port <parent-port-name> <trunk-name>
```

Example:

```
openstack network trunk create --parent-port primary-port my-trunk
```

Example output:

```
+----------------+--------------------------------------+
| Field          | Value                                |
+----------------+--------------------------------------+
| id             | 550e8400-e29b-41d4-a716-446655440000 |
| name           | my-trunk                             |
| parent_port_id | f47ac10b-58cc-4372-a567-0e02b2c3d479 |
| status         | DOWN                                 |
| sub_ports      |                                      |
+----------------+--------------------------------------+
```

> **Info**
>
> At this point, the trunk exists but is not attached to any server. The parent port is a standard Neutron port that will be referenced when creating the instance.

4. Create a Sub-Port

Create a Neutron port on each network you want to make accessible through the trunk:

```
openstack port create --network <network-name> <sub-port-name>
```

Example:

```
openstack port create --network network-1 sub-port-1
```

5. Add Sub-Port to the Trunk

Attach the sub-port to the trunk, specifying the segmentation type (`vlan`) and the segmentation ID matching the network's vlan tag:

```
openstack network trunk set \
  --subport port=`<sub-port-name>`,segmentation-type=vlan,segmentation-id=`<vlan-id>` \
  `<trunk-name>`
```

Example:

```
openstack network trunk set \
  --subport port=sub-port-1,segmentation-type=vlan,segmentation-id=100 \
  my-trunk
```

> **Warning**
>
> The behaviour of `segmentation-id` differs depending on the instance type:
>
> - **Bare metal:** the `segmentation-id` **must match** the segmentation ID of the network assigned to the sub-port. Neutron does not verify this value, but if it does not match, traffic will not reach the instance.
> - **Virtual machines:** the `segmentation-id` can be **any value** you choose. The hypervisor handles the translation between the sub-port vlan tag and the network's actual segmentation ID.

> **Info**
>
> To add more networks, repeat steps 4 and 5 for each additional network. For bare metal instances, use the matching `segmentation-id` of each network.

6. Verify the Trunk Configuration

Confirm the trunk is properly configured with all expected sub-ports:

```
openstack network trunk show <trunk-name>
```

Example:

```
openstack network trunk show my-trunk
```

Example output:

```
+----------------+-------------------------------------------------------------------------------------------+
| Field          | Value                                                                                     |
+----------------+-------------------------------------------------------------------------------------------+
| id             | 550e8400-e29b-41d4-a716-446655440000                                                      |
| name           | my-trunk                                                                                  |
| parent_port_id | f47ac10b-58cc-4372-a567-0e02b2c3d479                                                      |
| status         | DOWN                                                                                      |
| sub_ports      | [{"port_id": "...", "segmentation_id": 100, "segmentation_type": "vlan"}]                 |
+----------------+-------------------------------------------------------------------------------------------+
```

7. Deploy an Instance Using the Trunk

Create the instance referencing the **parent port**. OpenStack will configure the trunk during provisioning.

```
openstack server create \
  --image <image-name> \
  --flavor <flavor> \
  --port <parent-port-name> \
  --key-name <keypair-name> \
  <instance-name>
```

Bare metal example:

```
openstack server create \
  --image ubuntu-22.04 \
  --flavor baremetal \
  --port primary-port \
  --key-name my-keypair \
  --availability-zone "nova::88830859-5b16-4935-8f41-d381b754cbe5" \
  my-trunk-instance
```

Virtual machine example:

```
openstack server create \
  --image ubuntu-22.04 \
  --flavor m1.large \
  --port primary-port \
  --key-name my-keypair \
  my-trunk-instance
```

> **Warning**
> You **must** use `--port` (referencing the parent port) rather than `--nic net-id=...`. Using `--nic` would create a new port and bypass the trunk configuration entirely.

Summary of Steps

Step	Action	Command
1	List networks	`openstack network list`
2	Create parent port	`openstack port create --network <network-name> <parent-port-name>`
3	Create trunk	`openstack network trunk create --parent-port <parent-port-name> <trunk-name>`
4	Create sub-port	`openstack port create --network <network-name> <sub-port-name>`
5	Add sub-port to trunk	`openstack network trunk set --subport port=<sub-port-name>,segmentation-type=vlan,segmentation-id=<vlan-id> <trunk-name>`
6	Verify trunk	`openstack network trunk show <trunk-name>`
7	Deploy instance	`openstack server create --port <parent-port-name> --flavor <flavor> ...`

Instance Operating System Configuration

After deploying your instance, you need to configure **vlan sub-interfaces** inside the guest OS to access each network attached through the trunk sub-ports.

> **Warning**
>
> Automatic trunk configuration via cloud-init is **not possible**. OpenStack does not pass trunk metadata to the instance userdata. You must configure vlan sub-interfaces manually or through a post-deployment provisioning tool.

> **Warning**
>
> On **bare metal instances**, since the parent port is a dummy port with no effect on the network fabric, the base network interface will **not** have any network connectivity by default. All networks must be accessed through vlan sub-interfaces matching the `segmentation-id` assigned to each sub-port.
>
> On **virtual machines**, the base interface carries the parent network as untagged traffic. Only sub-port networks require vlan sub-interfaces.

1. Identify the Main Network Interface

Connect to your instance and identify the primary network interface:

```
ip link show
```

Look for the main interface (e.g., `ens3`, `ens21f0np0`, or `bond0` if LACP is configured). This is the physical interface carrying the trunk.

2. Create vlan Sub-Interfaces (Temporary)

For each sub-port, create a vlan sub-interface matching the `segmentation-id` you assigned. This is a non-persistent method for testing:

```
sudo ip link add link <main-interface> name <main-interface>.<vlan-id> type vlan id <vlan-id>
sudo ip link set <main-interface>.<vlan-id> up
sudo ip addr add <ip-address>/<cidr> dev <main-interface>.<vlan-id>
```

Example:

```
sudo ip link add link ens3 name ens3.100 type vlan id 100
sudo ip link set ens3.100 up
sudo ip addr add 192.168.1.10/24 dev ens3.100
```

> **Warning**
>
> This configuration will not survive a reboot. See the next step for a persistent configuration.

3. Persistent Configuration (Netplan Example)

For a persistent vlan sub-interface configuration using Netplan (Ubuntu/Debian with cloud-init), create a configuration file (e.g., `/etc/netplan/60-vlans.yaml`):

```
network:
  version: 2
  vlans:
    ens3.100:
      id: 100
      link: ens3
      addresses:
        - 192.168.1.10/24
    ens3.200:
      id: 200
      link: ens3
      addresses:
        - 10.0.0.10/24
```

Then apply the configuration:

```
sudo netplan apply
```

> **Info**
>
> If your instance uses LACP bonding (see LACP guide), replace `ens3` with your bond interface name (e.g., `bond0`). The vlan sub-interfaces then become `bond0.100`, `bond0.200`, etc.

4. Verify Connectivity

Check that your vlan sub-interfaces are up and have the correct IP addresses:

```
ip addr show <main-interface>.<vlan-id>
```

Then test connectivity:

```
ping <gateway-or-peer-on-vlan>
```

Example:

```
ip addr show ens3.100
ping 192.168.1.1
```

> **Tip**
>
> If the ping succeeds, your vlan sub-interface is correctly configured and the trunk is carrying traffic for the corresponding network.

Conclusion

You have successfully configured:

- **Neutron Trunk ports** at the OpenStack level, connecting an instance to multiple networks via vlan tagging;
- **vlan sub-interfaces** within the guest OS to access each network attached through trunk sub-ports;
- And verified **network connectivity** on each vlan.

Your instance can now communicate on multiple isolated networks through a single trunk configuration.

Go further

If you need training or technical assistance for the implementation of our solutions, contact your sales representative or click this link to request a quote and have your project analyzed by our Professional Services team experts.

Join our community of users.

How to configure a software RAID on a node

Find out how to configure and manage a software RAID on an OpenStack Ironic node in OPCP

Objective

This guide explains how to configure and manage a **software RAID** on an OpenStack Ironic baremetal node in your OPCP environment.

Software RAID allows you to create a redundancy configuration at the software level, without requiring a dedicated hardware RAID controller. This solution is particularly useful to improve the availability and storage performance of your instances.

This guide covers:

- Configuring software RAID via the Ironic agent interface
- Verifying the RAID configuration
- Best practices for managing software RAID

> **Warning**
>
> Software RAID must be configured **before** deploying an instance on the node. Once an instance has been deployed, changing the RAID configuration requires deleting the instance and reconfiguring the node. Creating or modifying a RAID array erases the data present on the disks used.
>
> Monitoring of the software RAID is the responsibility of the end customer: we do not have access to the deployed instance to manage monitoring. You should configure alerting (for example using the `mdadm --monitor` command) and integrate this monitoring into your existing monitoring tools.

Requirements

Before you begin, make sure you have the following:

- An active OPCP service.
- Access to **Configured OpenStack CLI** with the required permissions (`clouds.yaml` or environment variables).
- The **admin** role and/or nodes transferred into your project.
- An available node (status `available`) or a node in maintenance mode.
- A **Linux** (GNU/Linux) system image is required for the instance. This image must include the `mdadm` package or allow its installation. VMware appliances and Windows operating systems, for example, are not compatible with this procedure.
- Basic knowledge of OpenStack Ironic and baremetal node management.

Why use software RAID?

Software RAID provides several benefits in an OPCP environment:

- **Data redundancy**: Protection against data loss in the event of a disk failure.

- **Performance improvement**: Distribution of read/write operations across multiple disks.
- **Lower cost**: No need for a dedicated hardware RAID controller.

Instructions

1. Check the disks available on the node

Before configuring RAID, you must identify the disks available on your node.

List available nodes:

```
openstack baremetal node list
```

Check the node hardware properties:

```
openstack baremetal node show <node-id>
```

2. Enable maintenance mode

Before any configuration change, put the node in **maintenance mode**:

```
openstack baremetal node maintenance set <node-id> --reason "Software RAID configuration"
```

3. Supported RAID levels

Ironic supports several software RAID levels. The following values are accepted in the JSON configuration:

RAID level	JSON value	Description	Minimum number of disks
RAID 1	`"1"`	Mirroring	2

> **Warning**
>
> **Important constraint**: The first logical disk with `is_root_volume: true` **must be in RAID 1**. Other RAID levels (RAID 0, 5, 6, 10, etc.) are not allowed for the root volume.

Therefore, only RAID 1 can be used for instance deployment.

4. Configure software RAID

Ironic allows you to configure software RAID via the **agent** interface. This configuration is applied automatically when an instance is deployed.

4.1. Check and enable the agent interface if necessary

Before configuring RAID, check the RAID interface currently configured on the node:

```
openstack baremetal node show `<node-id>` -f json | jq '.raid_interface'
```

If the output is `null` or different from `"agent"`, enable the agent RAID interface:

```
openstack baremetal node set <node-id> --raid-interface=agent
```

4.2. Create the RAID configuration file

Create a JSON file containing the desired RAID configuration. The following example creates a RAID 1 (mirroring) array with two disks:

```
cat > /tmp/raid1.json <<EOF {
  "logical_disks": [{
    "controller": "software",
    "size_gb": "MAX",
    "raid_level": "1",
    "is_root_volume": true,
    "physical_disks": [
      {"size": "<1000"},
      {"size": "<1000"}
    ]
  }]
}
EOF
```

> **Info**
>
> The `"size": "<1000"` parameter in `physical_disks` automatically selects disks smaller than 1000 GB. You can adjust this value according to the size of your disks or use other selection criteria.

4.3. Apply the RAID configuration

Once the configuration file has been created, apply it to the node:

```
openstack baremetal node set `<node-id>` --target-raid-config /tmp/raid1.json
```

4.4. Check the RAID configuration

To check the RAID configuration applied on a node:

```
openstack baremetal node show `<node-id>` -f json | jq '.target_raid_config'
```

5. Disable maintenance mode

Once the RAID configuration is complete, disable maintenance mode:

```
openstack baremetal node maintenance unset `<node-id>`
```

6. Deploy an instance on the configured node

Once RAID has been configured, you can deploy an instance on the node:

```
openstack server create \
  --image `<image-name>` \
  --flavor `<flavor-id>` \
  --key-name `<keypair-name>` \
  --nic net-id=`<network-id>` \
  --availability-zone "nova::`<node-id>`" \
  `<instance-name>`
```

> **Tip**
>
> To make sure that your instance is deployed on the node configured with RAID, use the availability zone `nova::<node-id>`.

7. Check the RAID configuration after deployment

Once the instance is deployed, you can check the RAID configuration from within the instance:

Check RAID devices:

```
cat /proc/mdstat
```

Example output:

```
Personalities : [raid1] [raid10] [linear] [multipath] [raid0] [raid6] [raid5] [raid4]
md0 : active raid1 sda[0] sdb[1]
      500G 0 blocks super 1.2 [2/2] [UU]

unused devices: <none>
```

Check detailed information for a RAID device:

```
mdadm --detail /dev/md0
```

Disabling RAID

In some cases, you may need to disable RAID on a node. To do this, set the RAID interface to `no-raid` and clear the target RAID configuration.

1. Disable RAID interface

Set the RAID interface to `no-raid`:

```
openstack baremetal node set `<node-id>` --raid-interface=no-raid
```

2. Clear target RAID configuration

Clear the target RAID configuration:

```
openstack baremetal node set `<node-id>` --target-raid-config "{}"
```

> **Warning**
> Disabling RAID will erase all data on the disks used for the RAID configuration. Make sure to backup any important data before proceeding.

Summary of main commands

Action	Command
List nodes	`openstack baremetal node list`
Enable maintenance mode	`openstack baremetal node maintenance set <node-id>`
Check RAID interface	`openstack baremetal node show <node-id> -f json | jq '.raid_interface'`
Enable agent RAID interface	`openstack baremetal node set <node-id> --raid-interface=agent`
Apply RAID configuration	`openstack baremetal node set <node-id> --target-raid-config /tmp/raid1.json`
Check RAID configuration	`openstack baremetal node show <node-id> -f json | jq '.target_raid_config'`
Disable maintenance mode	`openstack baremetal node maintenance unset <node-id>`
Deploy an instance	`openstack server create --image <image-name> --flavor <flavor-id> --key-name <keypair-name> --nic net-id=<network-id> --availability-zone "nova::<node-id>" <instance-name>`
Check RAID status (from the instance)	`cat /proc/mdstat`
Disable RAID interface	`openstack baremetal node set <node-id> --raid-interface=no-raid`
Clear target RAID configuration	`openstack baremetal node set <node-id> --target-raid-config "{}"`

Best practices

- **Always configure RAID before deployment**: The configuration must be done on a node with no active instance.
- **Use maintenance mode**: Always put the node in maintenance before making any changes.
- **Regular backups**: RAID is not a backup solution; make regular backups of your data.
- **Monitor the RAID**: Set up RAID monitoring (for example with `mdadm --monitor`) and integrate alerts into your monitoring tools to quickly detect any degradation or disk failure.

Limitations and considerations

- **Performance**: Software RAID can have an impact on CPU performance compared to hardware RAID.
- **Compatibility**: Not all operating systems support all software RAID levels.

- **Rebuild**: Rebuilding a RAID array after replacing a disk can take a long time and consume resources.

Troubleshooting

Error	Cause	Solution
`Driver redfish does not support raid (disabled or not implemented). (HTTP 404)`	The RAID interface is not configured as `agent`	See the section 4.1 - Check and enable the agent interface if necessary
`Software RAID Configuration requires RAID-1 for the first logical disk`	The first logical disk with `is_root_volume: true` must be RAID 1	Use RAID 1 for the root volume. See the section 3 - Supported RAID levels

References

- OpenStack Ironic Documentation - RAID Configuration
- mdadm Documentation
- OPCP Guide - Node lifecycle

Go further

If you need training or technical assistance to implement our solutions, please contact your sales representative or click on this link to request a quote and have your project reviewed by our Professional Services experts.

Join our community of users.

How to see the node inventory

Find out how to access the OPCP node inventory

Objective

This guide explains how to view the hardware inventory, consumed resources, and available resources on an **OVHcloud On-Prem Cloud Platform (OPCP)** using:

- **Grafana**: real-time infrastructure monitoring
- **NetBox**: physical and logical inventory
- **OpenStack Horizon**: Ironic inventory via Horizon
- **OpenStack CLI**: Ironic inventory via CLI

> **Info**
>
> In OpenStack, a node corresponds to a physical server in the OPCP rack.
>
> In this guide, the term **node** is therefore used to refer to a physical server.

Prerequisites

- Being an administrator of the OPCP infrastructure and have access to the administration interface `admin.dashboard`.
- **OpenStack CLI configured** with the required permissions (`clouds.yaml` or environment variables).

Instructions

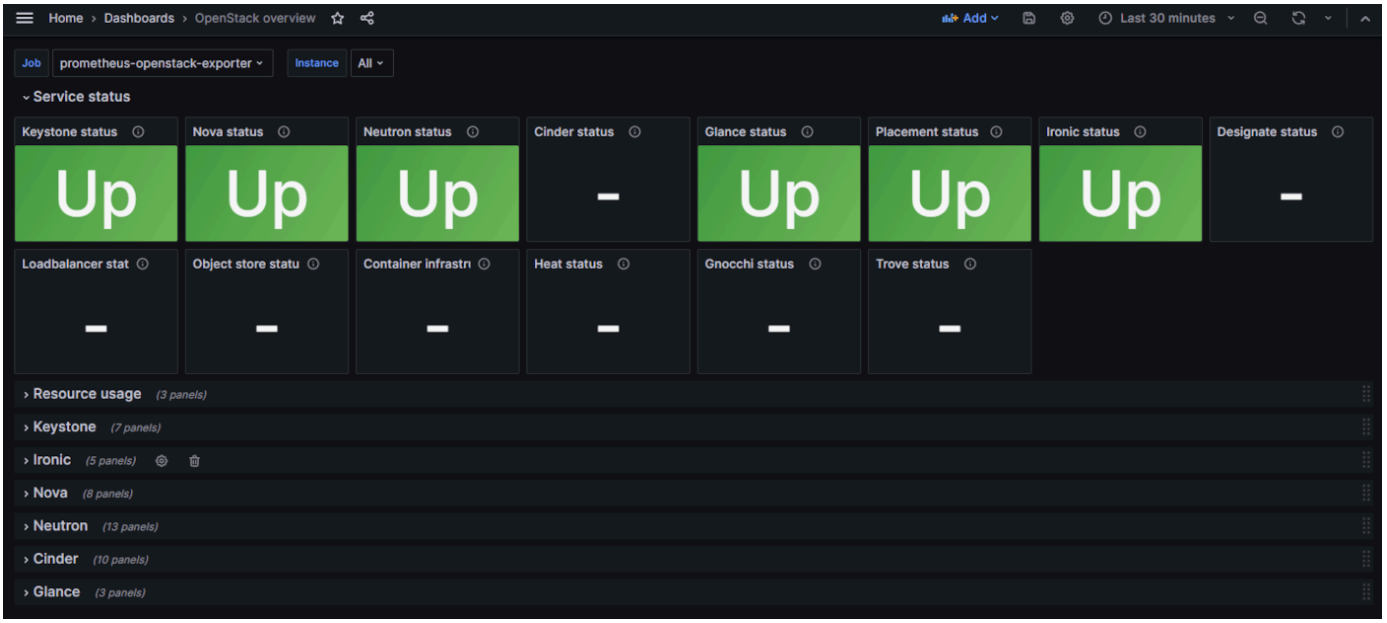
1. Inventory and monitoring via Grafana

Grafana provides a real-time monitoring view and allows you to easily track changes in your OPCP infrastructure.

1.1 Access the Home dashboard

1. Log in to the administration URL `admin.dashboard`.
2. Click **Grafana**.
3. Click **Dashboards** and search for the **OpenStack overview** dashboard.

You should see a dashboard similar to the following image:



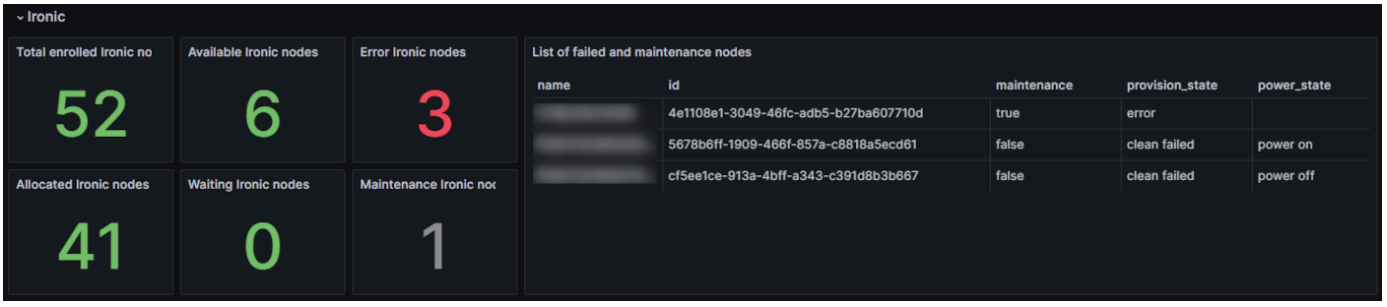
1.2 Understanding the OpenStack overview dashboard

The dashboard allows you to visualize:

- OpenStack service status: **Service status**
- Resource usage: **Resource usage**
- A detailed view of the Keystone service: **Keystone**
- A detailed view of the Ironic service: **Ironic**
- A detailed view of the Nova service: **Nova**
- A detailed view of the Neutron service: **Neutron**
- A detailed view of the Cinder service: **Cinder**
- A detailed view of the Glance service: **Glance**

1.3 Understanding and using the Ironic section

We will focus on the "Ironic" section, as this is where you can see the status of the different nodes.



This section allows you to view:

- Total number of nodes: **Total enrolled Ironic nodes**
- Number of available nodes not in maintenance: **Available Ironic nodes**
- Number of nodes in error: **Error Ironic nodes**
- List of nodes in error or maintenance: **List of failed and maintenance nodes**
- Number of allocated nodes: **Allocated Ironic nodes**
- Number of waiting nodes: **Waiting Ironic nodes**

- Number of nodes in maintenance: **Maintenance Ironic nodes**

2. Hardware and logical inventory with NetBox

NetBox is the reference tool for managing the physical, network, and logical inventory of your infrastructure.

The NetBox inventory is automatically populated; no manual action is required to add or modify nodes.

2.1 Access NetBox

1. Log in to the administration URL `admin.dashboard`.
2. Click **NetBox**.

2.2 View the node inventory

In **Devices**, you can see the complete list of equipment in your OPCP rack (nodes, network equipment, etc.).

You can:

- use the quick search to find an item by name;
- use **Filters** for advanced searches.

For example, you can list all non-production nodes by applying the following filters:

- **Status**: Decommissioned, Inventory, Failed, Planned
- **Role**: server

You can save the search to reuse it later.

3. OpenStack Horizon – Ironic UI

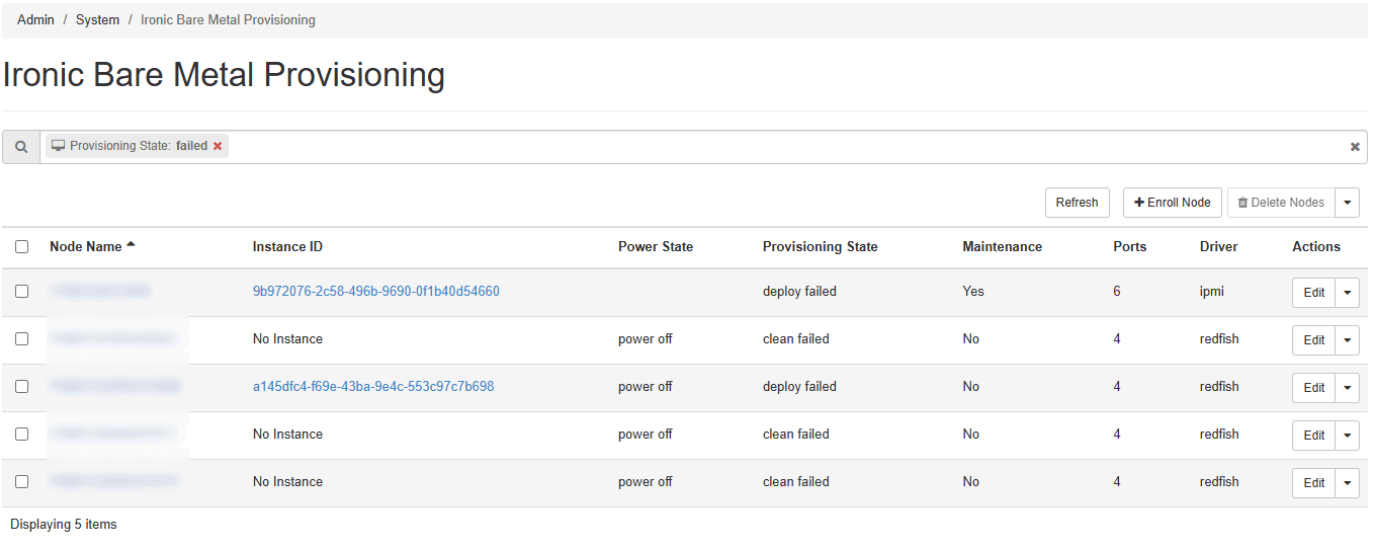
3.1 Access the Ironic interface

1. Log in to the administration URL `admin.dashboard`
2. Click **Horizon**
3. Click **Admin**
4. Click **System**, then **Ironic Bare Metal Provisioning**

3.2 List all nodes

In **Ironic Bare Metal Provisioning**, you can see all nodes.

You can filter nodes by their status, for example: `Provisioning State = failed`.



4. Resource status via OpenStack CLI

The OpenStack CLI allows you to obtain the **real-time** status of allocated and available resources.

4.1 List all nodes

```
openstack baremetal node list
```

Example output:

```
+--------------------------------------+---------------+--------------------------------------+-------------+-------
-------------+-------------+
| UUID                                 | Name          | Instance UUID                        | Power State |
Provisioning State | Maintenance |
+--------------------------------------+---------------+--------------------------------------+-------------+-------
-------------+-------------+
| 88830859-5b16-4935-8f41-d381b754cbe5 | SERVER-1      | None                                 | power off   |
available          | False       |
| 726bb7e9-3b20-4a44-99d7-2af747983781 | SERVER-2      | None                                 | power off   |
available          | False       |
| af711edf-a579-491e-b474-3c03639ed99b | SERVER-3      | 83b7083b-bbdd-4327-941c-ee91197818c5 | power on    | active
| False       |
+--------------------------------------+---------------+--------------------------------------+-------------+-------
-------------+-------------+
```

4.2 Filter nodes

You can filter results using **--provision-state** to view nodes in a specific state, or **--maintenance** to view nodes in maintenance.

All available filters can be found using `--help`.

Example output:

```
$ openstack baremetal node list --provision-state "clean failed"
+--------------------------------------+---------------+--------------+-------------+--------------------+---------
----+
| UUID                                 | Name          | Instance UUID | Power State | Provisioning State |
Maintenance |
+--------------------------------------+---------------+--------------+-------------+--------------------+---------
----+
| 37f96160-16cf-47e6-8bee-0ee42a59fafe | SERVER-1      | None          | power off   | clean failed       | False
|
| 5678b6ff-1909-466f-857a-c8818a5ecd61 | SERVER-2      | None          | power off   | clean failed       | False
|
| cf5ee1ce-913a-4bff-a343-c391d8b3b667 | SERVER-3      | None          | power off   | clean failed       | False
|
+--------------------------------------+---------------+--------------+-------------+--------------------+---------
----+
```

```
$ openstack baremetal node list --maintenance
+--------------------------------------+-------------+--------------------------------------+-------------+---------
-----------+-------------+
| UUID                                 | Name        | Instance UUID                        | Power State |
Provisioning State | Maintenance |
+--------------------------------------+-------------+--------------------------------------+-------------+---------
-----------+-------------+
| 4e1108e1-3049-46fc-adb5-b27ba607710d | SERVER-4    | 9b972076-2c58-496b-9690-0f1b40d54660 | None        | deploy
failed      | True        |
+--------------------------------------+-------------+--------------------------------------+-------------+---------
-----------+-------------+
```

4.3 Node details

To retrieve all information about a node, for example to identify the last error and understand why the deployment failed:

```
openstack baremetal node show `<node-id>`
```

Go further

If you need training or technical assistance for the implementation of our solutions, contact your sales representative or click this link to request a quote and have your project analyzed by our Professional Services team experts.

Join our community of users.

How to handle the NetBox rack elevation

Find out how the rack elevation is handled in NetBox for your OPCP infrastructure

Objective

NetBox is the reference tool for the physical and logical inventory of your **OVHcloud On-Prem Cloud Platform (OPCP)** infrastructure. Among its features, the **rack elevation** provides a visual, to-scale representation of a rack and the equipment installed in it.

This guide explains how the rack elevation is handled in NetBox and what information it gives you about your OPCP rack.

> **Info**
>
> The NetBox inventory is automatically populated from the OPCP control plane; the OPCP nodes are added and kept up to date without any manual action. Their **position** in the rack, however, is currently set manually (see section 4).
>
> In OpenStack, a node corresponds to a physical server in the OPCP rack. In this guide, the term **node** therefore refers to a physical server.

Prerequisites

- Being an administrator of the OPCP infrastructure and having access to the administration interface `admin.dashboard`.
- Knowing how to access the NetBox inventory (see How to see the node inventory).

Instructions

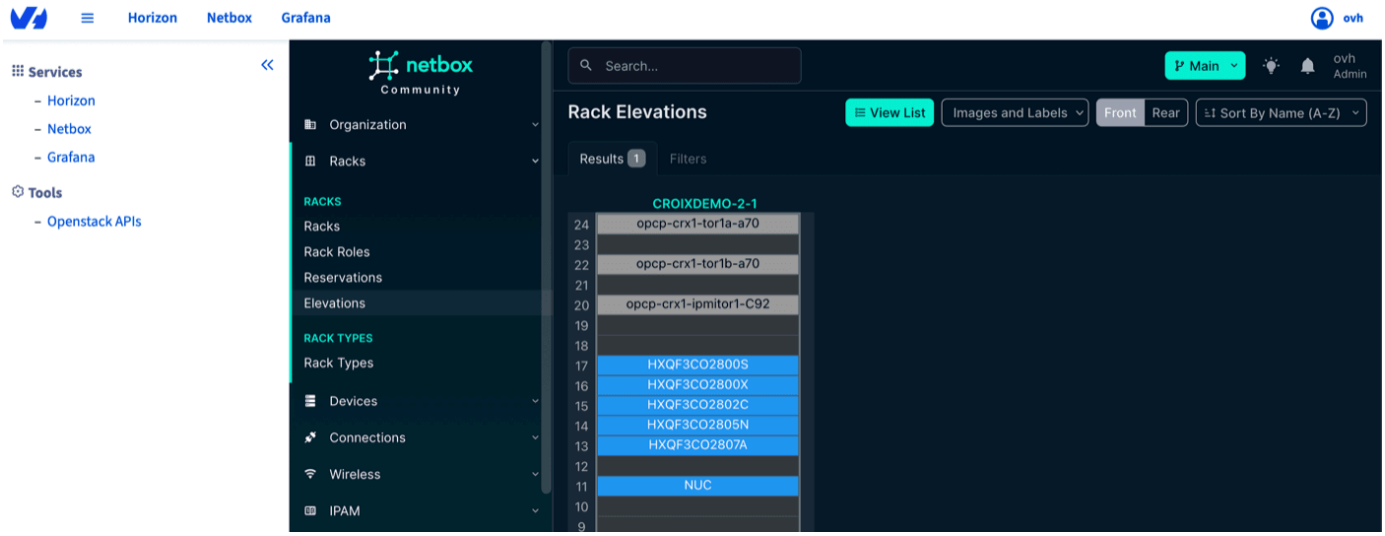
1. What is a rack elevation

In NetBox, a **rack** represents a physical equipment enclosure inside a data centre. Its capacity is measured in **rack units (U)** — a standard rack is typically between 42U and 48U high.

The **rack elevation** is a to-scale diagram of that rack. It shows, unit by unit, which device occupies which position, so you can understand the physical layout of your OPCP infrastructure without being on site.

2. Accessing the rack elevation in NetBox

1. Log in to the administration URL `admin.dashboard`.
2. Click **NetBox**. The rack elevation of your OPCP rack opens directly.



Alternative path: if you need to reach the elevation from another part of NetBox, open the **DCIM** section, select **Racks**, then select your OPCP rack to open its detail page and open the **Elevation** view.

3. Reading the elevation view

The elevation view conveys several pieces of information at a glance:

- **Front and rear faces**: the elevation can be displayed from the front or the rear of the rack, so you can see equipment that is mounted on either side.
- **Device position and height**: each device is drawn at the rack unit (U) where it is mounted and spans the number of units it physically occupies.
- **Unit numbering**: rack units are numbered along the side of the elevation. Depending on the rack configuration, numbering can run from the bottom up or from the top down.
- **Role-based colouring**: devices are coloured according to their role (for example servers versus network equipment), making it easy to distinguish equipment types.
- **Free space**: empty rack units are shown as available space, giving you a quick view of the remaining capacity.

4. How node positions are handled

Today, the position of each node in the rack elevation is set **manually**. This is the standard approach: an administrator sets a device's position by editing the device in NetBox and assigning it a rack and a rack unit (U). A position set this way is persistent: it is not changed automatically, and remains until you change it yourself.

You can also add **custom devices** to the rack elevation — equipment that is not managed as an OPCP node, such as third-party appliances you install yourself. These are created and positioned manually in NetBox in the same way. Before placing a custom device, contact OVHcloud support to confirm which rack unit(s) you may use for it, so that it does not conflict with units reserved for the platform.

> **Info**
>
> **Automatic positioning from templates is on the way.** OVHcloud is working on automatic placement of nodes based on their cabling. Once this feature is available, OVHcloud will provide **position templates** that map a given set of switch-port connections to a specific rack unit (U), per **rack type** (which depends on how the platform was installed). During the Ironic → NetBox synchronisation, a node whose cabling matches a template entry will then be placed automatically at the corresponding rack unit, and this automatic position will take precedence over manual changes. Where no template matches the rack type, manual positioning will remain the fallback. Custom devices, which have no Ironic cabling, will always continue to be positioned manually.

5. What the rack elevation tells you

For an OPCP infrastructure, the rack elevation helps you to:

- **Locate a node physically** in the rack for on-site operations such as maintenance or cabling.
- **Assess capacity** by seeing which units are occupied and which are free.
- **Map the physical and logical views together** by cross-referencing a device shown in the elevation with its node inventory entry and its OpenStack/Ironic status.

To go further with the inventory and node states, see:

- How to see the node inventory
- Node lifecycle

References

- NetBox documentation - Racks
- NetBox documentation - Rack Types

Go further

For training or technical assistance implementing our solutions, contact your sales representative or visit our Professional Services page to request a quote and have your project analysed by our experts.

Join our community of users.

How the Ironic to NetBox synchronisation works

Find out how OPCP nodes are imported from Ironic into NetBox and what the synchronisation keeps up to date

Objective

On an **OVHcloud On-Prem Cloud Platform (OPCP)**, the physical and logical inventory shown in NetBox is not maintained by hand: it is populated automatically from OpenStack Ironic, which is the source of truth for the bare-metal nodes of your platform.

This guide explains how the Ironic to NetBox synchronisation works and what information it keeps up to date.

> **Info**
>
> The NetBox inventory is automatically populated from the OPCP control plane; no manual action is required to add, update, or remove nodes.
>
> In OpenStack, a node corresponds to a physical server in the OPCP rack. In this guide, the term **node** therefore refers to a physical server.

Prerequisites

- Being an administrator of the OPCP infrastructure and having access to the administration interface `admin.dashboard`.
- Knowing how to access the NetBox inventory (see How to see the node inventory).

Instructions

1. Overview of the synchronisation

The synchronisation runs automatically and reconciles the NetBox inventory with the current state of Ironic. On each run, it:

1. Fetches the list of nodes and their network ports from Ironic.
2. Creates or updates the matching **devices** in NetBox.
3. Keeps interfaces, MAC addresses, and cabling to the switches in sync.
4. Computes and updates the **rack position** of each node (see section 3).
5. Decommissions devices that no longer exist in Ironic (see section 4).

Because NetBox is kept in sync with Ironic, it reflects the same reality as the OpenStack/Ironic view of your nodes. The node states described in Node lifecycle are therefore mirrored into NetBox.

2. What the synchronisation imports

For each node, the synchronisation imports and keeps up to date:

- **Device identity**: the device name, its site and rack.

- **Device type and manufacturer**: derived from the node's hardware model and vendor reported by Ironic.
- **Identifiers**: the Ironic node UUID is stored both as the device **asset tag** and in a dedicated `baremetal_node_id` field, so a NetBox device can always be mapped back to its Ironic node.
- **BMC information**: the BMC MAC address and BMC address are stored on the device.
- **Status**: mapped from the Ironic provision state.
- **Interfaces and MAC addresses**: one interface per Ironic port, with its MAC address.
- **Cabling**: cables between each node interface and the corresponding switch port, based on the link information reported by Ironic.
- **Rack position**: the rack unit (U) of the node, when it can be determined (see section 3).

3. Rack position and elevation

The rack position imported by the synchronisation is what drives the **rack elevation** view in NetBox. The position is not entered by hand: it is computed from the node's cabling and from the position templates that OVHcloud provides for your rack type.

How the position is determined, when automatic placement applies, and when manual positioning is kept as a fallback are explained in detail in the How to handle the NetBox rack elevation guide.

4. Decommissioning of removed nodes

When a node that previously existed in Ironic is no longer returned by Ironic, the synchronisation does not delete its device in NetBox. Instead, it:

- sets the device status to **Decommissioning**;
- clears its rack position and face, so it no longer appears in the rack elevation;
- removes the cables attached to its interfaces.

This keeps a trace of the device in NetBox while making it clear that it is no longer part of the active inventory.

References

- How to see the node inventory
- Node lifecycle
- How to handle the NetBox rack elevation
- NetBox documentation - Devices

Go further

For training or technical assistance implementing our solutions, contact your sales representative or visit our Professional Services page to request a quote and have your project analysed by our experts.

Join our community of users.

How to use Terraform

Find out how to generate an Application Credential from Horizon and automate the deployment of your OPCP resources with Terraform

Objective

Terraform is an open source **Infrastructure as Code (IaC)** tool developed by HashiCorp. It allows you to describe and provision your infrastructure declaratively from configuration files written in *HashiCorp Configuration Language* (HCL).

Since the **OPCP** offer is based on **OpenStack**, you can use the **Terraform OpenStack provider** to automate the deployment of your resources: instances, networks, volumes, key pairs, etc.

> **Info**
>
> This guide also applies to **OpenTofu**, the open source fork of Terraform maintained by the Linux Foundation. OpenTofu is compatible with the HCL syntax and Terraform providers: simply replace the `terraform` command with `tofu` in the examples below.

This guide details the steps required to generate an *Application Credential* from Horizon, configure Terraform and deploy a first server on your OPCP infrastructure.

Requirements

- An active OPCP service.
- A **user account** with sufficient rights to log in to Horizon on the OPCP offer.
- Terraform installed (version >= 1.0) or OpenTofu on your workstation.
- An **SSH key pair** on your local workstation to access your instance.
- A **private network** previously created in your OPCP project (see the guide How to install an instance from the Horizon interface).

Table of Contents

Instructions

1. Creating an Application Credential from Horizon

To allow Terraform to communicate with your OPCP infrastructure, you need to generate an **Application Credential** pair (`id` / `secret`) from the Horizon interface. This mechanism avoids using your Keycloak credentials directly and provides dedicated authentication for your automation workflows, with a permissions scope limited to the current project.

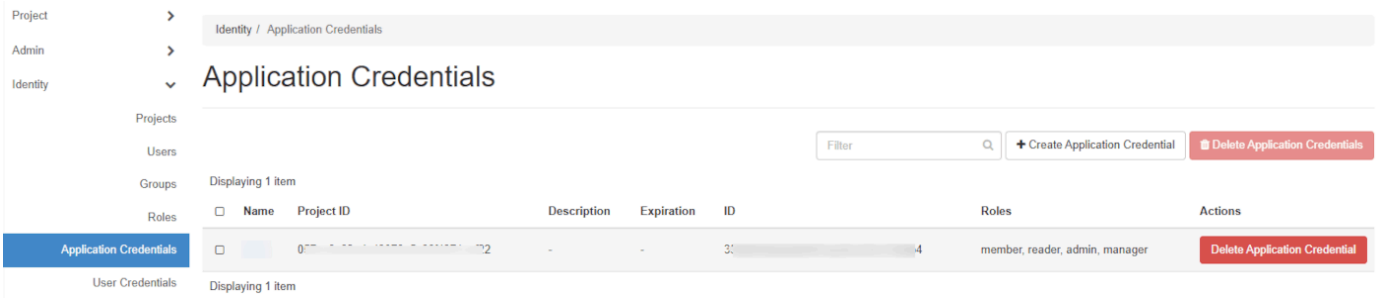
> **Warning**
>
> An `Application Credential` is automatically deleted when the user who created it is revoked. To avoid any loss of access in an automation workflow, do not generate an Application Credential from a nominative or easily revocable user account; use a dedicated account instead.

Logging in to Horizon

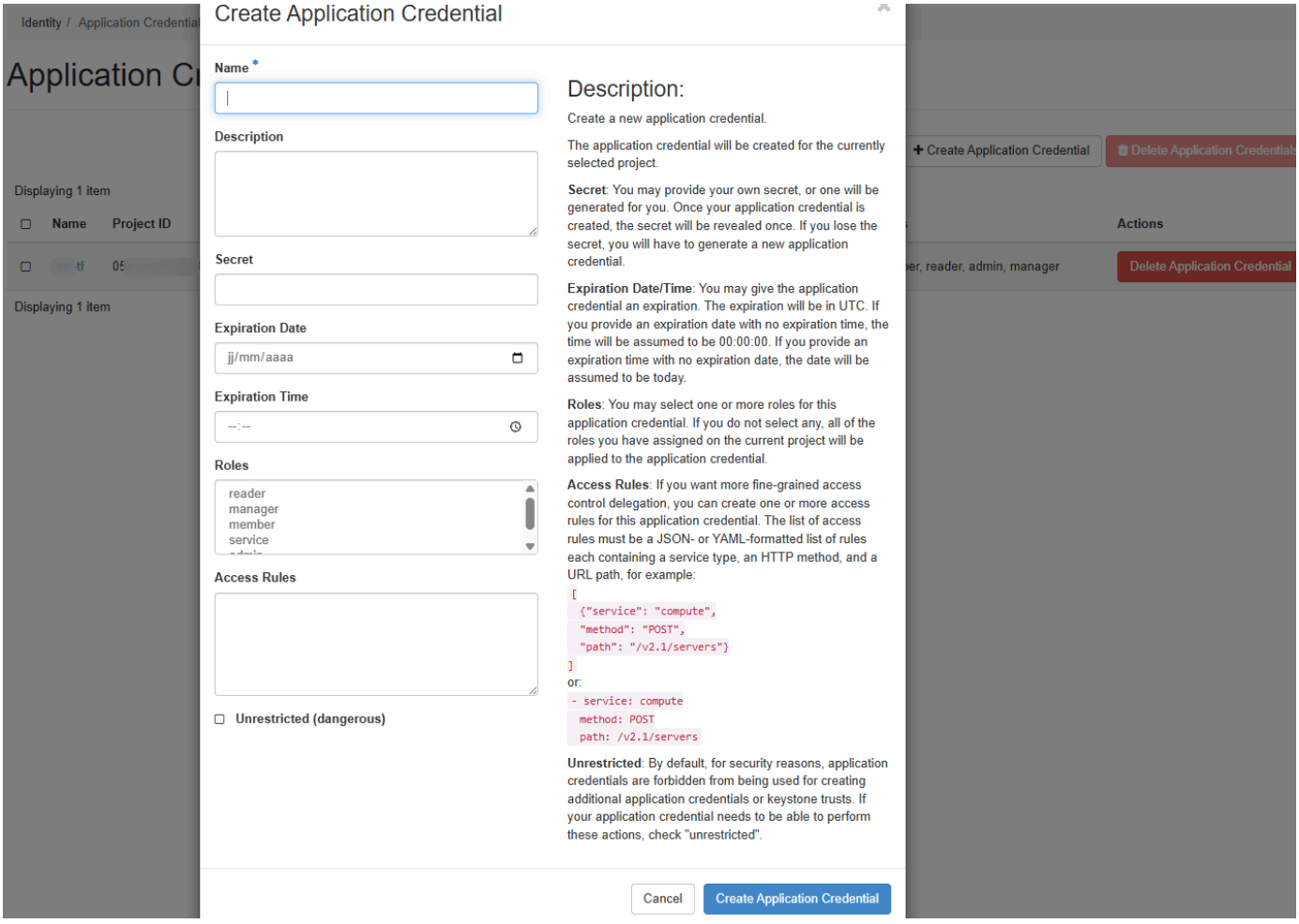
Log in to the **Horizon** interface of your OPCP environment, then select the **project** in which you want to deploy your resources via Terraform. For more information, refer to the Getting started with your OPCP guide.

Creating the Application Credential

In the left-hand menu, click on **Identity**, then on **Application Credentials**.



Click on **+ Create Application Credential**.

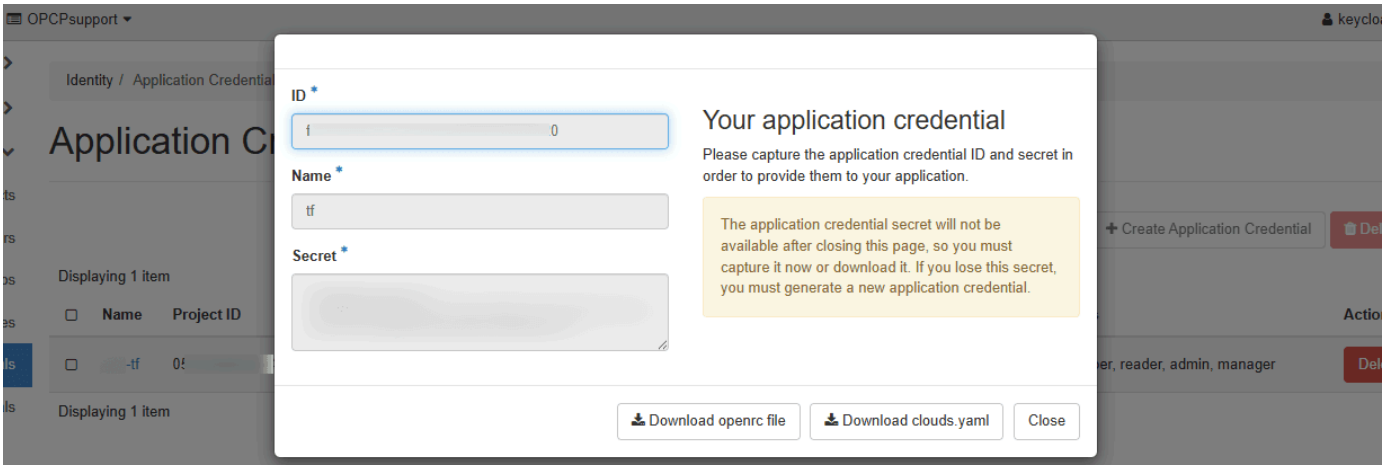


Fill in the following fields:

Field	Description
Name	Enter a name for your Application Credential (e.g.: *terraform-cred*).
Description	Optional. Add a description if needed.
Secret	Optional. If left blank, a secret will be generated automatically.
Expiration Date / Expiration Time	Optional. Set an expiration date.
Roles	Select the roles to associate with the credential (e.g.: *member*, *admin*).
Access Rules	Optional. Allows you to restrict the authorised actions.
Unrestricted	Leave unchecked to limit the possible actions.

Click on **Create Application Credential**.

> **Warning**
>
> Once the window is closed, the **secret will no longer be accessible**. Download the `clouds.yaml` or `openrc` file offered by Horizon, or copy the `id` and `secret` values to a secure location.



> **Info**
>
> For the rest of this tutorial, **download the `openrc` file**: it will be used in the next step to authenticate Terraform against your OPCP infrastructure.

2. Preparing the Terraform environment

Creating the working directory

Create a directory dedicated to your Terraform project:

```
mkdir terraform-opcp && cd terraform-opcp
```

Defining the OpenStack provider

In a file named `provider.tf`, add the following lines:

```
terraform {
  required_version = ">= 1.0"
  required_providers {
    openstack = {
      source  = "terraform-provider-openstack/openstack"
      version = ">= 3.0.0"
    }
  }
}

provider "openstack" {

}
```

No parameters are required in the **`provider`** block: the OpenStack provider automatically retrieves the authentication information from the `OS_*` environment variables.

Loading OpenStack environment variables

When you created your Application Credential, Horizon offered to download a `clouds.yaml` or `openrc` file. The simplest approach is to source this `openrc.sh` file in your shell before running Terraform:

```
source openrc.sh
```

> **Info**
>
> The `openrc.sh` file exports `OS_AUTH_URL`, `OS_REGION_NAME`, `OS_APPLICATION_CREDENTIAL_ID` and `OS_APPLICATION_CREDENTIAL_SECRET`, among others. These variables will automatically be used by the Terraform OpenStack provider.

Initialisation

Download the OpenStack provider plugins:

```
terraform init
```

3. Creating a server

In a `main.tf` file, declare the resources required to create an instance attached to an existing private network:

```
# Variables related to the instance
variable "instance_name" {
  description = "Name of the instance that will be created in OPCP"
  type        = string
  default     = "instance-terraform-name"
}

variable "image_name" {
  description = "Name of the image to use for the instance (e.g.: Debian 12 OPCP)"
  type        = string
  default     = "Debian 12 OPCP"
}

variable "flavor_name" {
  description = "Baremetal flavor defining the hardware resources of the instance (e.g.: scale-1, scale-2, etc.)"
  type        = string
  default     = "scale-1"
}

variable "network_name" {
  description = "Name of the existing network to which the instance will be attached"
  type        = string
  default     = "<your-network-name>"
}

variable "key_name" {
  description = "Name of the SSH key pair to associate with the instance"
  type        = string
  default     = "terraform-key"
}

# Creating an instance
resource "openstack_compute_instance_v2" "terraform_instance" {
  name        = var.instance_name
  image_name  = var.image_name
  flavor_name = var.flavor_name
  key_pair    = var.key_name

  network {
    name = var.network_name
  }

  lifecycle {
    # Prevents Terraform from detecting drift when the base image is updated
    ignore_changes = [
      image_name
    ]
  }

  # Useful metadata
  metadata = {
    environment = "production"
    managed_by  = "terraform"
  }

  # Longer timeouts (baremetal provisioning is slow)
  timeouts {
    create = "60m"
    delete = "30m"
  }
}

# Useful outputs after creation
output "instance_id" {
```

```
  value = openstack_compute_instance_v2.terraform_instance.id
}

output "instance_ip" {
  value = openstack_compute_instance_v2.terraform_instance.access_ip_v4
}
```

> **Info**
>
> The names of available images, flavors and networks can be listed from Horizon or with the OpenStack CLI (`openstack image list` , `openstack flavor list` , `openstack network list`). To configure the CLI, refer to the guide How to use the API and get credentials.

Reviewing the plan

Before any deployment, preview the actions that will be performed:

```
terraform plan
```

Applying the configuration

Deploy the instance with the following command:

```
terraform apply
```

Confirm with `yes` when Terraform asks you to. Once the creation is complete, the instance is visible in the Horizon interface, in the **Compute** > **Instances** section.

4. Additional node-level configurations (RAID, LACP)

Some configurations must be applied **on the baremetal node** before deploying the instance and are not covered by the Terraform OpenStack provider. They require **admin** Ironic rights (or nodes transferred to your project) and must be performed via the OpenStack CLI.

> **Info**
>
> **Software RAID**: to configure software RAID on a baremetal node, refer to the guide How to set up software RAID on a node. The `target_raid_config` attribute is not exposed by the `openstack_baremetal_node_v1` resource. This operation must be performed **before** the `terraform apply` that deploys the instance, and the configured node should then be targeted via `availability_zone = "nova::<node-id>"` .

> **Info**
>
> **LACP / bonding**: to aggregate several network interfaces of a node, refer to the guide How to set up LACP on a node. Baremetal port and bonding configuration cannot be managed declaratively by the Terraform OpenStack provider. This operation must be performed **before** the `terraform apply` that deploys the instance.

5. Removing the infrastructure

To delete all resources created via Terraform:

```
terraform destroy
```

> **Warning**
> `terraform destroy` does not reset the RAID or LACP configurations applied to the node. To remove them, follow the dedicated section of the corresponding guide using the OpenStack CLI.

6. Troubleshooting

Issue	Possible cause	Solution
`Authentication failed`	Wrong `application_credential_id` or `secret`	Check the credentials in Horizon (**Identity** > **Application Credentials**).
`Could not find image`	The image name does not match	List the available images via `openstack image list` or from Horizon.
`No suitable endpoint could be found`	Wrong `auth_url` URL or non-existent region	Check the Keystone URL and the region in **Project** > **API Access** .
`Network not found`	Private network missing in the project	Create a private network beforehand from Horizon (**Network** > **Networks**).

7. References

- Official Terraform documentation
- Official OpenTofu documentation
- Terraform OpenStack provider
- openstack_compute_instance_v2 resource
- OpenStack Application Credentials

Go further

For training or technical assistance implementing our solutions, contact your sales representative or visit our Professional Services page to request a quote and have your project analyzed by our experts.

Join our community of users.

How to update the backup S3 buckets

Find out how to change the S3 object storage buckets used by the OPCP platform backups

Objective

Your **OVHcloud On-Prem Cloud Platform (OPCP)** stores its platform backups in an external **S3[1]-compatible object storage**. These backups cover the critical components required to restore the control plane, namely:

- the internal **databases** (`db`),
- the **persistent volumes** (`pv`),
- the **Terraform state** (`tfState`),
- the **KMS auto-unsealer shares** (`kmsShares`).

You may need to change the object storage buckets or endpoint used for these backups in several situations: migrating to a new S3 endpoint or region, moving to a different object storage provider, renaming buckets, or adjusting the retention policy.

This guide explains how to update the backup S3 buckets in the OPCP configuration, redeploy the control plane, and re-initialize the **Velero** / **Kopia** backup repositories so they point to the new object storage location.

> **Warning**
> This procedure modifies the backup configuration of the platform and re-initializes the backup repositories. Existing backups stored in the **previous** buckets are **not** migrated automatically. Make sure the previous backups are no longer needed (or have been copied over) before starting, and perform this operation during a maintenance window.

Requirements

- An operational OPCP infrastructure.
- Administrative access to the OPCP controller with the `opcp-cli` , `opcp-diag` , `kubectl` , `flux` , `tfctl` and `velero` tools installed and configured.
- The connection details of the **target** S3 object storage: endpoint URL, region, and the names of the buckets to use for the backups.
- A maintenance window, as the control plane is redeployed during the operation.

Instructions

1. Running a pre-flight diagnostic

Before making any change, check that the platform is healthy. The `opcp-diag` tool runs a series of platform checks:

```
opcp-diag run -p
```

Only proceed once the diagnostic passes. This gives you a clean baseline to compare against once the operation is complete.

2. Updating the backup configuration

Open the OPCP configuration in your editor:

```
opcp-cli config edit
```

Locate the `backups` section and update it with the new object storage endpoint and bucket names:

```
...
backups:
  endpoint:
    region: yourRegion
    url: "url.of.your.s3.service"
  db:
    enabled: true
    retention: "30d"
    bucket: "opcp01-backups-db-region"
  pv:
    enabled: true
    retention: "720h0m0s"
    bucket: "opcp01-backups-pv-region"
    tfState:
      retention: "720h0m0s"
    kmsShares:
      retention: "720h0m0s"
...
```

The fields to review are:

- `endpoint.region` / `endpoint.url`: the region and URL of the target S3-compatible service.
- `db.bucket`: the bucket storing the database backups, with its `retention` (here `30d`).
- `pv.bucket`: the bucket storing the persistent volume backups, with its `retention` (here `720h0m0s`, i.e. 30 days).
- `pv.tfState.retention` / `pv.kmsShares.retention`: the retention applied to the Terraform state and KMS shares backups.

> **Info**
>
> Retention values can be expressed either as a day-based duration (for example `30d`) or as a Go duration string (for example `720h0m0s`, which is also 30 days). Keep the format used by the existing configuration.

Save and close the file. The command only **updates** the stored configuration; it does **not** apply it yet. The change is applied in the next step with `opcp-cli deploy`. The CLI reports when the edit is done:

```
✅ Command 'edit' completed in 1m12.081606602s
```

3. Applying the new configuration

Apply the updated configuration:

```
opcp-cli deploy
```

This command does **not** redeploy the entire control plane: it only runs the convergence needed to apply your change, that is, it updates the Velero backup configuration and the database backup configuration.

The operation can take several minutes. Wait until you see the confirmation:

```
...
✅ Deployment of control plane terminated
✅ Command 'deploy' completed in 10m19.19421549s
```

You can follow the deployment in real time in a separate terminal by tailing the CLI log:

```
tail -F /var/log/opcp-cli/opcp-cli.log | jq .msg
```

4. Waiting for reconciliation

The platform relies on **Flux** (GitOps reconciliation) and **Terraform** to converge to the new desired state. Wait until every resource has reconciled.

List any resources that are not yet ready:

```
flux get all -A --status-selector ready=false
flux get all -A --status-selector ready=unknown
```

Check the state of the Terraform reconciliations:

```
tfctl get
```

Wait for these commands to report no pending or failing resources. If a resource stays blocked, you may occasionally need to restart the relevant pods — be patient, reconciliation can take a while.

5. Verifying the backup storage location

Once reconciliation is complete, confirm that Velero now points to the new bucket and that the storage location is `Available`:

```
velero backup-location get
```

```
NAME              PROVIDER   BUCKET/PREFIX               PHASE       LAST VALIDATED                  ACCESS MODE
DEFAULT
backup-location   aws        opcp01-backups-velero-region   Available   2026-07-07 12:30:41 +0000 UTC   ReadWrite
true
```

The `PHASE` must be `Available`. It can take a few minutes after the deployment before the storage location is validated and becomes `Available`, so wait a moment and run the command again if needed. If it stays in another state, re-check the endpoint and bucket configuration from step 2 and the credentials of the S3 service.

6. Re-initializing the Velero backup repositories

The **Kopia** backup repositories used by Velero are bound to the previous object storage location. To make Velero recreate them against the new buckets, you need to delete the existing repositories and restart the Velero components.

First, list the current backup repositories:

```
kubectl -n velero get backuprepositories -A
```

```
NAMESPACE   NAME                                  AGE    REPOSITORY TYPE
velero      glance-backup-location-kopia-jqj4d    215d   kopia
velero      ironic-backup-location-kopia-w46h4    215d   kopia
velero      kms-backup-location-kopia-jxdvz       215d   kopia
velero      monitoring-backup-location-kopia-hqjnt   215d   kopia
```

Delete all of them:

```
kubectl -n velero delete backuprepositories --all
```

```
backuprepository.velero.io "glance-backup-location-kopia-jqj4d" deleted from velero namespace
backuprepository.velero.io "ironic-backup-location-kopia-w46h4" deleted from velero namespace
backuprepository.velero.io "kms-backup-location-kopia-jxdvz" deleted from velero namespace
backuprepository.velero.io "monitoring-backup-location-kopia-hqjnt" deleted from velero namespace
```

Confirm that no repository remains:

```
kubectl -n velero get backuprepositories -A
```

```
No resources found
```

Restart the Velero deployment and the `node-agent` daemonset so they re-create the repositories against the new buckets:

```
kubectl -n velero rollout restart deployment/velero
kubectl -n velero rollout restart daemonset/node-agent
```

> **Info**
>
> A `Warning` message may be displayed when restarting these workloads. It is expected and can be safely ignored.

Check that the Velero and `node-agent` pods have restarted and are `Running`:

```
kubectl get pod -n velero
```

```
NAME                                                READY   STATUS      RESTARTS   AGE
...
node-agent-p92tl                                    1/1     Running     0          62s
node-agent-pkhwn                                    1/1     Running     0          58s
node-agent-qk6j2                                    1/1     Running     0          60s
...
velero-5f7fd4c8b8-rqprb                             1/1     Running     0          68s
```

7. Triggering a first backup for each schedule

The platform defines several backup schedules. List them to confirm they are enabled:

```
velero get schedule
```

```
NAME                              STATUS    CREATED                         SCHEDULE    BACKUP TTL   LAST BACKUP
SELECTOR                                    PAUSED
kms-autounsealer-shares-backup    Enabled   2025-12-03 13:30:21 +0000 UTC   0 * * * *   720h0m0s     34m ago
kms-autounsealer-shares-backup=enabled   false
pvs-backup                        Enabled   2025-12-03 13:30:21 +0000 UTC   0 * * * *   720h0m0s     34m ago
backup=true                                 false
tfstate-backup                    Enabled   2025-12-03 13:30:21 +0000 UTC   0 * * * *   720h0m0s     34m ago
tfstate=true                                false
```

Rather than waiting for the next scheduled run, trigger an immediate backup from each schedule. The `--wait` flag makes the command block until the backup completes.

Back up the KMS auto-unsealer shares:

```
velero create backup --from-schedule kms-autounsealer-shares-backup --wait
```

Back up the persistent volumes:

```
velero create backup --from-schedule pvs-backup --wait
```

Back up the Terraform state:

```
velero create backup --from-schedule tfstate-backup --wait
```

Each command ends with a `Backup completed with status: Completed.` message. For example:

```
Backup request "pvs-backup-20260707123616" submitted successfully.
Waiting for backup to complete. You may safely press ctrl-c to stop waiting - your backup will continue in the
background.
.....................................................................................................
.................................
Backup completed with status: Completed. You may check for more information using the commands `velero backup
describe pvs-backup-20260707123616` and `velero backup logs pvs-backup-20260707123616`.
```

8. Verifying the backups and the recreated repositories

Confirm that all three backups completed without errors:

```
velero get backup
```

```
NAME                                          STATUS      ERRORS   WARNINGS   CREATED
EXPIRES   STORAGE LOCATION   SELECTOR
kms-autounsealer-shares-backup-20260707123604   Completed   0        0          2026-07-07 12:36:04 +0000 UTC   29d
backup-location    kms-autounsealer-shares-backup=enabled
pvs-backup-20260707123616                       Completed   0        0          2026-07-07 12:36:16 +0000 UTC   29d
backup-location    backup=true
tfstate-backup-20260707123915                   Completed   0        0          2026-07-07 12:39:15 +0000 UTC   29d
backup-location    tfstate=true
```

Check that Velero has recreated the Kopia backup repositories against the new buckets (they should now show a recent `AGE`):

```
kubectl -n velero get backuprepositories
```

```
NAME                                    AGE     REPOSITORY TYPE
glance-backup-location-kopia-6sdt6      5m20s   kopia
ironic-backup-location-kopia-hnbjh      5m18s   kopia
kms-backup-location-kopia-567f6         5m17s   kopia
monitoring-backup-location-kopia-kvmjv  5m15s   kopia
```

9. Backing up the databases

The internal databases are managed by **CloudNativePG** and backed up separately, to the `barmanObjectStore` that now points to the new bucket. Trigger a database backup for each database.

> **Info**
>
> The commands below use the `placement-db` database in the `placement` namespace as an example. Repeat them for the other databases, adapting the database name and its namespace accordingly.

```
kubectl cnpg backup placement-db -n placement
```

```
backup/placement-db-20260707124646 created
```

Confirm that it reaches the `completed` state:

```
kubectl get backups -n placement
```

```
...
placement-db-20260707124646   5s      placement-db   barmanObjectStore   completed
```

10. Checking that data has been written to the new buckets

The previous steps report that the backups completed from the platform's point of view. As an additional check, verify that the objects were actually created in the **new** S3 buckets.

Using your usual S3 client or object storage interface, browse each backup bucket and confirm that new objects have appeared since the operation. You should see recently dated objects corresponding to the database, persistent volume, Terraform state and KMS shares backups you triggered. This confirms that the backups are actually stored on the new object storage.

11. Final verification

Run the diagnostic again to confirm the platform is healthy after the change:

```
opcp-diag run -p
```

Finally, confirm that the scheduled backups now run automatically on their new location. After the next scheduled run, `velero get backup` should show the newly created backups alongside the ones you triggered manually:

```
velero get backup
```

```
NAME                                          STATUS      ERRORS   WARNINGS   CREATED
EXPIRES   STORAGE LOCATION   SELECTOR
kms-autounsealer-shares-backup-20260707130021   Completed   0        0          2026-07-07 13:00:21 +0000 UTC   29d
backup-location    kms-autounsealer-shares-backup=enabled
kms-autounsealer-shares-backup-20260707123604   Completed   0        0          2026-07-07 12:36:04 +0000 UTC   29d
backup-location    kms-autounsealer-shares-backup=enabled
pvs-backup-20260707130021                       Completed   0        0          2026-07-07 13:00:22 +0000 UTC   29d
backup-location    backup=true
pvs-backup-20260707123616                       Completed   0        0          2026-07-07 12:36:16 +0000 UTC   29d
backup-location    backup=true
tfstate-backup-20260707130021                   Completed   0        0          2026-07-07 13:00:30 +0000 UTC   29d
backup-location    tfstate=true
tfstate-backup-20260707123915                   Completed   0        0          2026-07-07 12:39:15 +0000 UTC   29d
backup-location    tfstate=true
```

Likewise, the database backups continue automatically:

```
kubectl get backups -n placement
```

```
...
placement-db-20260707124646   20m     placement-db   barmanObjectStore   completed
placement-db-20260707130000   6m54s   placement-db   barmanObjectStore   completed
```

Your OPCP platform backups now target the new S3 buckets.

Go further

For training or technical assistance implementing our solutions, contact your sales representative or visit our Professional Services page to request a quote and have your project analyzed by our experts.

Join our community of users.

[1]: S3 is a trademark of Amazon Technologies, Inc. OVHcloud's service is not sponsored by, endorsed by, or otherwise affiliated with Amazon Technologies, Inc.

Security

IAM rights management

Find out how to manage users, roles, and access rights on your On-Prem Cloud Platform with Keycloak

Objective

This guide explains how to manage IAM (Identity and Access Management) rights and access on your **On-Prem Cloud Platform (OPCP)** through **Keycloak**, which centralises authentication to the platform's application services.

Requirements

- A delivered and operational On-Prem Cloud Platform service
- Access to the Keycloak interface with an account holding the `it_admin` role or higher
- Familiarity with the guide Getting started with your OPCP

Instructions

Authentication architecture

Authentication on the **OPCP** relies on two distinct layers:

- **Control plane access**: SSH access to the controllers that make up the platform infrastructure, provided at service delivery
- **Service access**: centralised in **Keycloak** (Realm Master), which serves as the single entry point for:
  - the **Dashboard** (unified service dashboard)
  - the **OpenStack Horizon** graphical interface
  - the **OpenStack APIs** (Keystone, Nova, Neutron, Glance, Ironic, etc.)
  - the monitoring tools: **Grafana**, **Netbox**, **Prometheus**

Control plane access

When your **OPCP** is made available, you receive credentials to connect via SSH to the various controllers that make up the platform's control plane.

The access scope of the administrator account provided to you varies depending on the chosen management mode.

OVHcloud-managed mode

In managed mode, the administrator account provided to you has extended rights to run the platform administration tools:

- `opcp-cli`: command-line tool for platform administration
- `opcp-diag`: control plane diagnostic tool

The deployment and updates of the control plane are handled by OVHcloud.

Non-managed mode

In non-managed mode, the administrator account provided to you has **full privileges** over the control plane, allowing you to:

- Deploy the control plane
- Update the control plane
- Access all administration tools

> **Warning**
> In non-managed mode, OVHcloud does not handle control plane administration operations. You are responsible for deployment and updates.

Initial credentials

Two types of credentials are provided when your OPCP is initialised:

Linux administrator account: used for SSH connections to the controllers. By default, **password authentication is disabled** on the controllers. During the installation of your OPCP, an SSH public key of your choice can be provided to OVHcloud to be deployed and allow you first access. A password is also provided, useful only for direct server access via the console in the event that SSH access is unavailable.

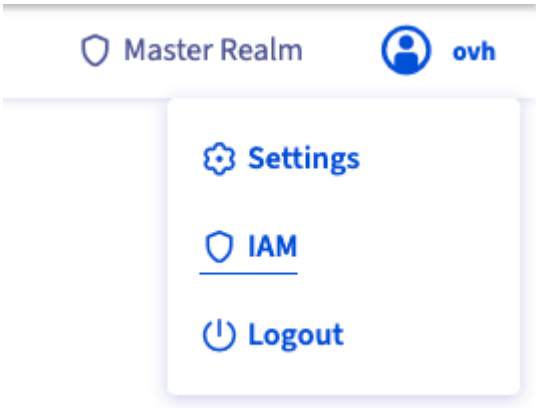
Keycloak administrator account: provided in both managed and non-managed modes, this account has full administration rights over the Realm Master. A temporary password is assigned, which you must **change at first login**.

> **Info**
> These initial credentials are separate from the user accounts you will subsequently create in Keycloak for your teams. Store them securely and change the temporary passwords immediately after delivery.

Accessing the Keycloak administration interface

Two methods are available to access the Keycloak administration interface of your **OPCP**.

Via the Dashboard: log in to `admin.dashboard.<domainname>`, then click on your name in the top right corner. A menu appears with an **IAM** link that redirects you directly to the Keycloak interface.



Direct access: you can also access Keycloak directly via `admin.keycloak.<domainname>`.

> **Info**
>
> Replace `<domainname>` with the domain name provided at the delivery of your OPCP.

Role hierarchy

The IAM model of the **OPCP** is based on four distinct roles, from the most restricted to the most privileged. These roles are a starting point — Keycloak gives you full flexibility to compose your own roles and assign them to user groups to match your organisation's needs.

> **Info**
>
> You can create custom roles in Keycloak by combining available permissions, then assign them to user groups. Refer to the official Keycloak documentation for advanced role and group management.

> **Warning**
>
> This section reflects OPCP version **3.0.6** and newer.

The `reader` role

Supervision and audit profile. By default, this role has read-only access and no administration rights.

It allows you to:

- View your own account settings in Keycloak
- Access OpenStack project resources according to the project attributes configured in Keycloak

> **Info**
>
> The Keycloak `reader` role can be granted any OpenStack role — `reader` (read-only), `member` (read-write) or `admin` — through project attributes configured on the Keycloak account or group. Without a project attribute, no OpenStack resource is visible.

The `dc_operator` role

Datacenter operator. This role is intended for teams responsible for network and physical infrastructure.

It provides the following additional access compared to `reader`:

- Access **Grafana** in read-only mode
- Access **Netbox** as an operator (create and modify network resources)

> **Info**
>
> The `dc_operator` has no Keycloak user administration rights. Like the `reader`, it holds no OpenStack rights by default: they require project attributes on the user or their group. It is not given the Dashboard's OpenStack iframe by default either — when you grant it OpenStack access through project attributes, grant that iframe as well.

The `it_admin` role

Main platform operator. This role is intended for IT teams who manage OPCP resources on a daily basis.

It provides the following additional access compared to `dc_operator`:

- Manage Realm Master users in Keycloak (create accounts, assign rights)
- Access **Grafana** in edit mode (modify dashboards)
- Access **OpenStack** with full rights (`reader`, `member`, `admin`)

> **Warning**
>
> The `it_admin` role does not allow modification of global Keycloak realm settings (password policy, identity federation, etc.). These actions are reserved for `master_admin`.

The `master_admin` role

Platform administrator. This role has full rights, including complete administration of the Keycloak Realm Master.

It provides the following additional access compared to `it_admin`:

- Modify the Realm Master configuration (security policy, federation, etc.)
- Access **Grafana** in admin mode
- Administer **Netbox** with full rights, including the `admin` role

> **Warning**
>
> The `master_admin` role is assigned by OVHcloud at service delivery. Its use should be limited to platform administration operations that cannot be delegated to an `it_admin`.

Rights matrix

> **Warning**
>
> This matrix reflects OPCP version **3.0.6** and newer.

The table below summarises the access for each role across the integrated applications.

Legend: ✅ Allowed | ❌ Not allowed | 🟡 Configuration required (subject to project attributes)

Application	Permission	`reader`	`dc_operator`	`it_admin`	`master_admin`
Keycloak	View own account settings	✅	✅	✅	✅
Keycloak	Manage realm users	❌	❌	✅	✅
Keycloak	Administer realm (policy, federation)	❌	❌	❌	✅
OpenStack	reader (view)	🟡	🟡	✅	✅
OpenStack	member (edit)	🟡	🟡	✅	✅
OpenStack	admin	🟡	🟡	✅	✅
Grafana	view (dashboards)	❌	✅	✅	✅
Grafana	edit (dashboards)	❌	❌	✅	✅
Grafana	admin (dashboards)	❌	❌	❌	✅
Netbox	reader	❌	✅	✅	✅
Netbox	operator	❌	✅	✅	✅
Netbox	admin	❌	❌	❌	✅
Prometheus	view	✅	✅	✅	✅
Dashboard	Keycloak — account settings	✅	✅	✅	✅
Dashboard	Keycloak — user administration	❌	❌	✅	✅
Dashboard	OpenStack (iframe)	✅	❌	✅	✅
Dashboard	Grafana (iframe)	❌	✅	✅	✅
Dashboard	Netbox (iframe)	❌	✅	✅	✅

> **Info**
>
> The `reader` and `dc_operator` roles' OpenStack access requires **project attributes** to be configured on the user or their group in Keycloak. Without a configured project attribute, no OpenStack resource is accessible. Any OpenStack role can technically be granted this way, but `member` is recommended: the `admin` role is not limited to a single project.

Creating a user and assigning a role

Creating a user

In the Keycloak interface, make sure you are connected to the **Realm Master**, then go to **Users** > **Add user**.

Fill in the following fields:

- **Username**: the user's login identifier

- **Email**: email address (recommended)
- **First name / Last name**: user's full name

Click **Create**, then open the **Credentials** tab to set a temporary password. Enable the **Temporary** option to force the user to change it at their first login.

Assigning a role to a user

In the user's profile, open the **Role mapping** tab, then click **Assign role**.

In the filter, select **Filter by realm roles**, then search for and select the desired role (`reader`, `dc_operator`, `it_admin` or `master_admin`). Click **Assign**.

> **Info**
>
> A user can be assigned multiple roles. The effective permissions are the union of the rights from all roles assigned to them.

> **Warning**
>
> The default role assigned to a new Keycloak user is always `reader`. Although the Keycloak interface allows you to change this default role at the realm level, this configuration is managed by Terraform on the **OPCP** and will be systematically overwritten during a reconciliation.

Assigning a role to a group

Assign roles to **groups** rather than to individual users to simplify rights management at scale.

In Keycloak, go to **Groups** > **Create group**, name the group, then open the **Role mapping** tab to assign the desired role.

Then add users to the group from their user profile via **Groups** > **Join group**.

Configuring OpenStack rights via Keycloak attributes

For the `reader` and `dc_operator` roles, access rights to OpenStack projects are defined directly in the **user attributes** (or group attributes) in Keycloak.

Adding a project attribute to a user

In the Keycloak interface, navigate to the relevant user, then open the **Attributes** tab.

Add a new attribute with the following values:

- **Key**: `project`
- **Value**: a JSON object describing the project and role to assign

```
{
    "domain": {
        "name": "Default"
    },
    "name": "project-name",
    "roles": [
        {
            "name": "reader"
        }
    ]
}
```

> **Info**
>
> For the `reader` and `dc_operator` roles, any OpenStack role can be assigned via project attributes, though `member` is recommended over `admin`, which is not limited to a single project. The `it_admin` and `master_admin` roles rely on a project attribute too, but theirs is set by default, so no configuration is required.

Assigning multiple projects to a user

You can add **multiple** `project` **attributes** to the same user or group. They will automatically be merged into a `projects` attribute in the token sent to Keystone.

Configuring attributes on a group

The same configuration can be applied to a **Keycloak group**. All members of the group will automatically inherit the project attributes defined on that group.

In Keycloak, go to **Groups**, select the desired group, then fill in the `project` attributes in the same way as for an individual user.

> **Info**
>
> Attributes defined on a group and those defined directly on the user are merged. A user can therefore benefit from projects from multiple groups in addition to their own attributes.

Keycloak as the single source of truth

We recommend not creating users directly in OpenStack, to keep Keycloak as the **single source of truth** for platform accounts. Any account created directly in OpenStack would bypass the lifecycle managed by Keycloak and would not benefit from automatic cleanup or centralised rights management.

Service accounts for automation

For automation needs (Terraform, OpenTofu, scripts, CI/CD pipelines, etc.), we recommend creating a **dedicated account in Keycloak** with permissions scoped to the automation's requirements, then generating **OpenStack application credentials** associated with that account.

Application credentials allow your automation tools to drive OpenStack without depending on a personal user's credentials. A user connected to OpenStack with their Keycloak account can create their own application credentials from the Horizon interface, via **Identity** > **Application Credentials** > **Create Application Credentials**.

> **Info**
>
> If the associated Keycloak account is deleted, the OpenStack application credentials will be automatically revoked. Avoid linking critical automation to a personal account.

For more details on configuring Keycloak authentication with the OpenStack CLI and obtaining your credentials, refer to the guide How to use the APIs and obtain the credentials.

Automatic account cleanup

The platform reacts to user deletion events in Keycloak: as soon as an account is removed from the Realm Master, the associated OpenStack users and **application credentials** are automatically deleted.

> **Warning**
>
> Deleting an account in Keycloak immediately triggers the revocation of the associated OpenStack application credentials. Consider this impact before deleting any account, particularly if automations or scripts rely on those credentials.

Advanced Keycloak configuration

All official Keycloak features are available on your **OPCP**. For any advanced configuration (password policies, fine-grained role management, client configuration, etc.), refer to the official Keycloak documentation.

Multi-factor authentication (MFA / OTP)

We strongly recommend enabling **multi-factor authentication** on your platform accounts, particularly for the `it_admin` and `master_admin` roles. Keycloak natively supports the TOTP (Time-based One-Time Password) protocol, compatible with any authentication application supporting this open standard, such as FreeOTP.

MFA configuration is done at the realm authentication flow level. Refer to the official Keycloak documentation to set up a post-login flow with mandatory or conditional OTP based on roles.

Identity federation and Identity Providers

Keycloak supports federating external directories (Active Directory, LDAP) or connecting third-party Identity Providers (SAML, OIDC). For example, to set up federation with an Active Directory, refer to the dedicated section of the official Keycloak documentation.

> **Warning**
>
> When setting up Keycloak federation or adding an Identity Provider pointing to an external endpoint, you must allow the corresponding network flows in the OPCP configuration file via the `keycloakAllowedEndpoints` variable.

```
keycloakAllowedEndpoints:
    - host: "myactivedirectory.corp.com"
      port: 636
    - host: "keycloak.corp.com"
      port: 443
```

Each entry corresponds to an external endpoint that Keycloak must be authorised to reach. Add as many entries as needed depending on your configuration.

Automation with OpenTofu / Terraform

User, group and role management in Keycloak can be automated using Infrastructure as Code tools such as **OpenTofu** or **Terraform**, via the official Keycloak provider.

This allows you to manage your access rights in a declarative, reproducible and version-controlled way.

Example resource to create a user:

```
resource "keycloak_user" "john_doe" {
  realm_id = "master"
  username = "john.doe"
  enabled  = true

  email      = "john.doe@corp.com"
  first_name = "John"
  last_name  = "Doe"

  initial_password {
    value     = "changeme"
    temporary = true
  }
}
```

For all available resources (users, groups, roles, federation, etc.), refer to the Keycloak provider documentation.

Listing all rights

Via opcp-diag (consolidated view)

To get a complete view of all accounts and their rights on the platform, connect via SSH to a controller and run:

```
opcp-diag audit
```

This tool displays all accounts declared in **Keycloak**, on the **controllers**, and across the various tools that make up the control plane.

Available options

Option	Description
`-o, --format`	Output format: `json` (default) or `text`
`--only`	Restrict the audit to one or more comma-separated types

The available audit types for `--only` are:

Type	Description
`keycloak_master`	Accounts and rights configured in the Keycloak Realm Master
`openstack`	OpenStack users and roles
`linux`	Accounts present on the controllers
`netbox`	Accounts in Netbox, the platform's CMDB
`nog`	Accounts in NOG, the component that drives network device configuration to apply Neutron networks

For example, to audit only Keycloak and OpenStack:

```
opcp-diag audit --only keycloak_master,openstack
```

For a human-readable text output:

```
opcp-diag audit --format text
```

Manipulating the JSON output

The default output is in **JSON** format, making it easy to manipulate and filter with tools like `jq`. For example:

```
# Export the full result to a dated file
opcp-diag audit > audit-$(date +%Y%m%d).json

# Audit only Keycloak and filter on a specific user
opcp-diag audit --only keycloak_master | jq '.keycloak_master.users[] | select(.username == "john.doe")'
```

> **Info**
>
> `opcp-diag audit` is the recommended method for auditing your platform's access. Unlike the OpenStack CLI, it covers all accounts present on the platform, regardless of their connection history.

Via the OpenStack CLI

To list active role assignments in OpenStack:

```
openstack role assignment list --names
```

To list the roles available on the platform:

```
openstack role list
```

> **Warning**
>
> These commands only return users who have **already authenticated** at least once via OpenStack. Users present in Keycloak who have never made an OpenStack connection will not appear in these results.

Go further

For training or technical assistance implementing our solutions, contact your sales representative or visit our Professional Services page to request a quote and have your project analyzed by our experts.

Join our community of users.

CloudStore

Getting started with your CloudStore

Find out how to log in, manage accounts, deploy services, and configure your CloudStore

Objective

This guide shows you how to log in to your CloudStore management interface, understand its key concepts, and perform initial operations such as creating accounts and deploying services.

The **CloudStore** is a high-level infrastructure framework deployed on top of On-Premise Cloud Platform (**OPCP**). It provides core services to help cloud providers deploy and manage cloud-native solutions for their customers through a marketplace-based platform.

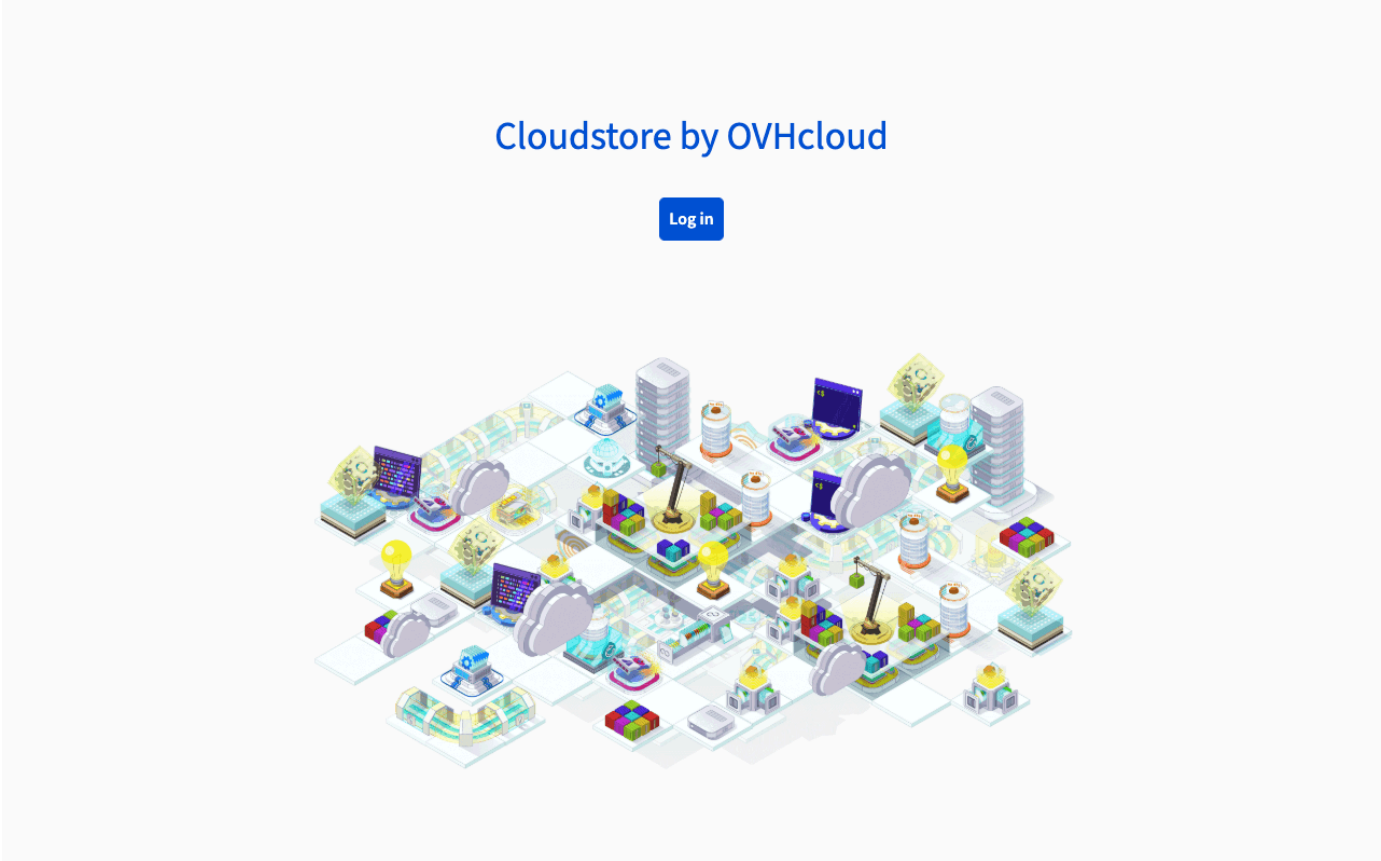
Requirements

- The **URL** of your CloudStore management interface, provided during service delivery.
- Login credentials (username and password) provided during service delivery. These credentials are managed through **Keycloak**, the identity and access management solution used by the platform.

Instructions

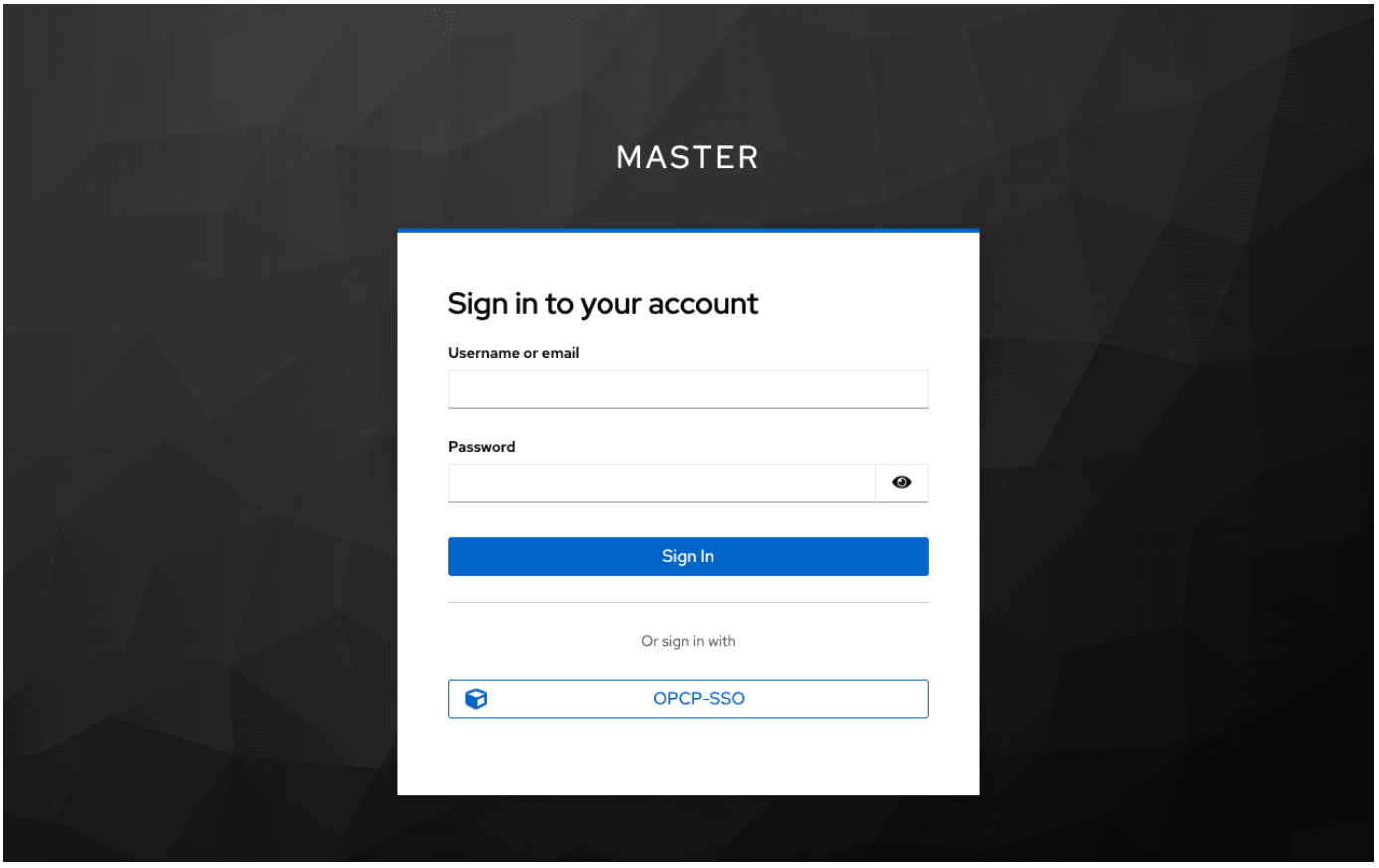
Logging in to CloudStore

Navigate to the **URL** provided for your CloudStore instance. You will be presented with a login page.



Click the login button to be redirected to the **Keycloak** authentication page. You have two options to authenticate:

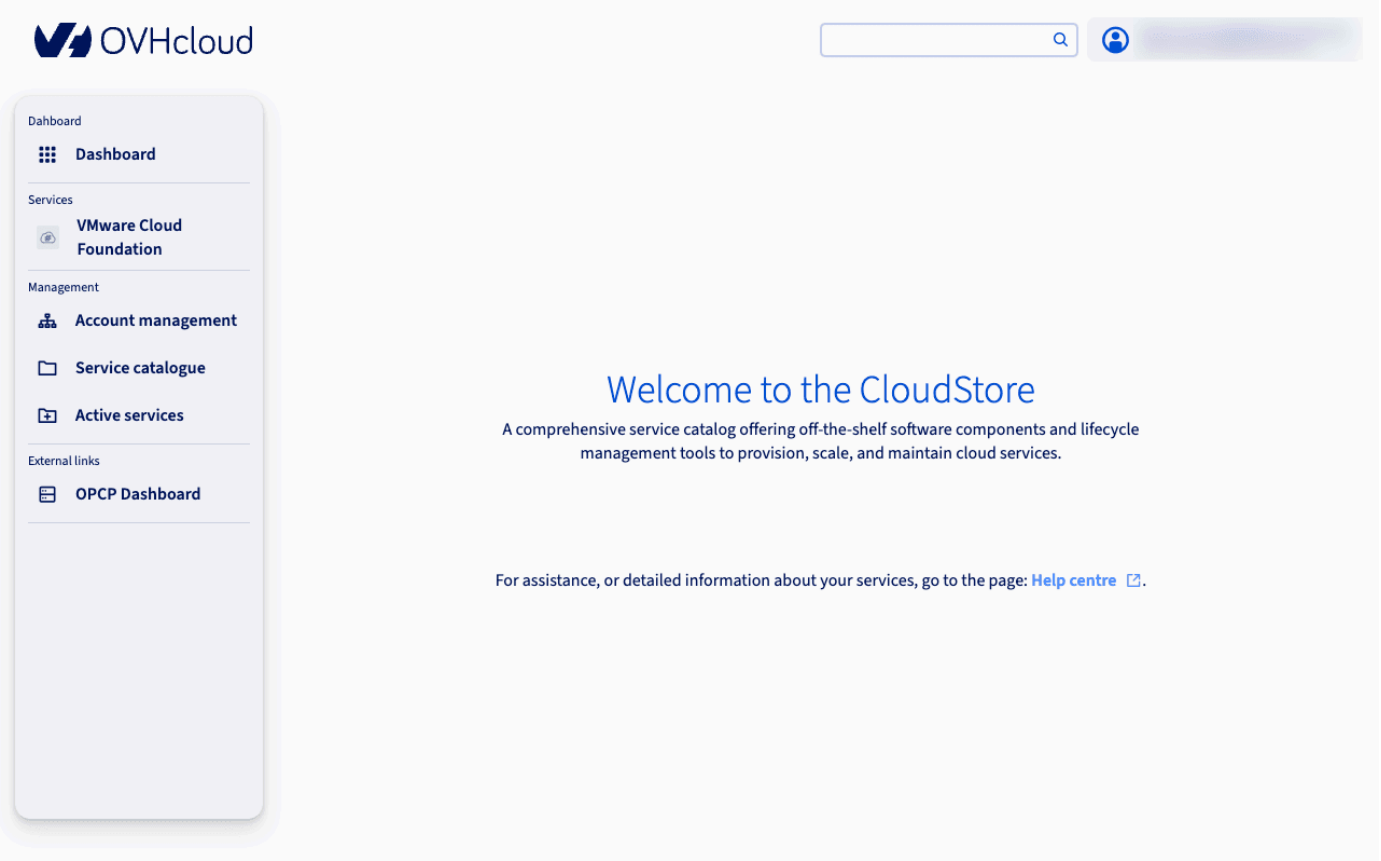
- Enter your credentials directly in the Keycloak login form.
- Use the **OPCP-SSO** button to authenticate through the federated OPCP Core Keycloak. This option allows users already registered in OPCP Core to log in without managing a separate set of credentials.



Once authenticated, the interface obtains an OIDC access token that is used for all subsequent operations.

CloudStore dashboard

After logging in, you will be redirected to the CloudStore dashboard.



From the dashboard, you can access the following key areas:

- **Service catalog**: Browse and activate available services (e.g., VCF).
- **Accounts**: Create and manage tenant accounts for your customers or internal teams.
- **Controllers and Apps**: Deploy and manage the infrastructure components of each service.
- **IAM management**: Configure users and permissions within Keycloak.

Key concepts

Before using the CloudStore, it is important to understand its main concepts.

Personas

The CloudStore distinguishes between two audiences:

- **Cloud providers** (IT admins, Service admins): OVHcloud customers who operate the platform. They provision infrastructure, deploy services, and manage accounts.
- **Cloud users** (Landing Zone Manager users): Customers of cloud providers who consume the services deployed for them through the **Landing Zone**.

Persona	Responsibilities
IT admin	Create accounts, activate services, deploy apps
Service admin	Manage a specific service, deploy its controller and apps
Account admin	Manage the configuration of their account
Landing Zone Manager user	Access apps through the Landing Zone

The controller/app pattern

Each service in CloudStore follows a **controller/app** architecture:

- The **controller** (controlplane) is deployed first. It manages the lifecycle of apps and handles cross-tenant orchestration. A single controller can manage multiple apps across different accounts.
- **Apps** (dataplane) are workloads deployed for a specific account. Each app is isolated to one account and cannot be shared between accounts.

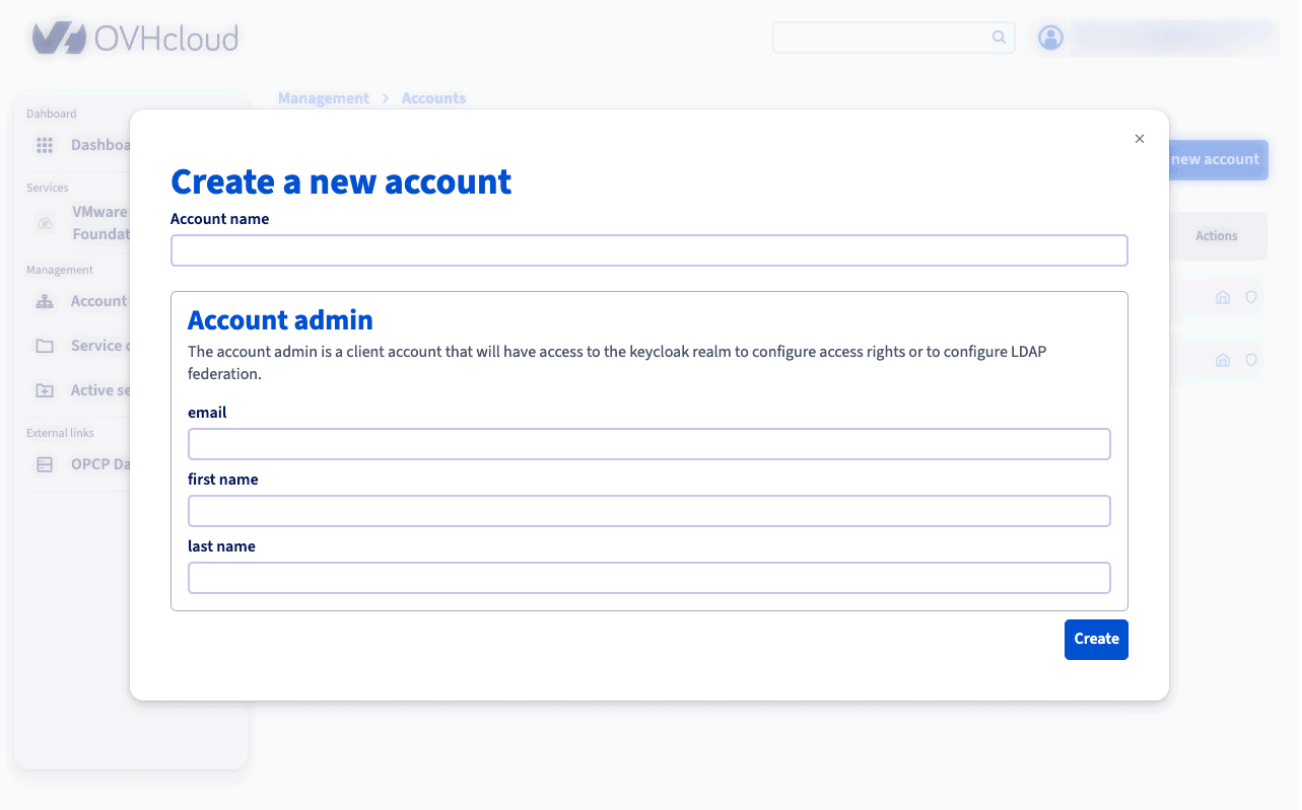
This pattern allows cloud providers to offer the same service to multiple customers, each getting their own isolated app instance while sharing the same controller infrastructure.

Creating an account

An **account** represents a company or a department. Each account gets its own Keycloak realm, providing complete IAM isolation. Accounts must be created before deploying apps, as apps are always scoped to an account.

To create an account:

1. From the CloudStore dashboard, navigate to the **Accounts** section.
2. Click on **Create an account**.
3. Fill in the required fields: account name, admin full name, and admin email address.



4. Submit the form.

The platform automatically:

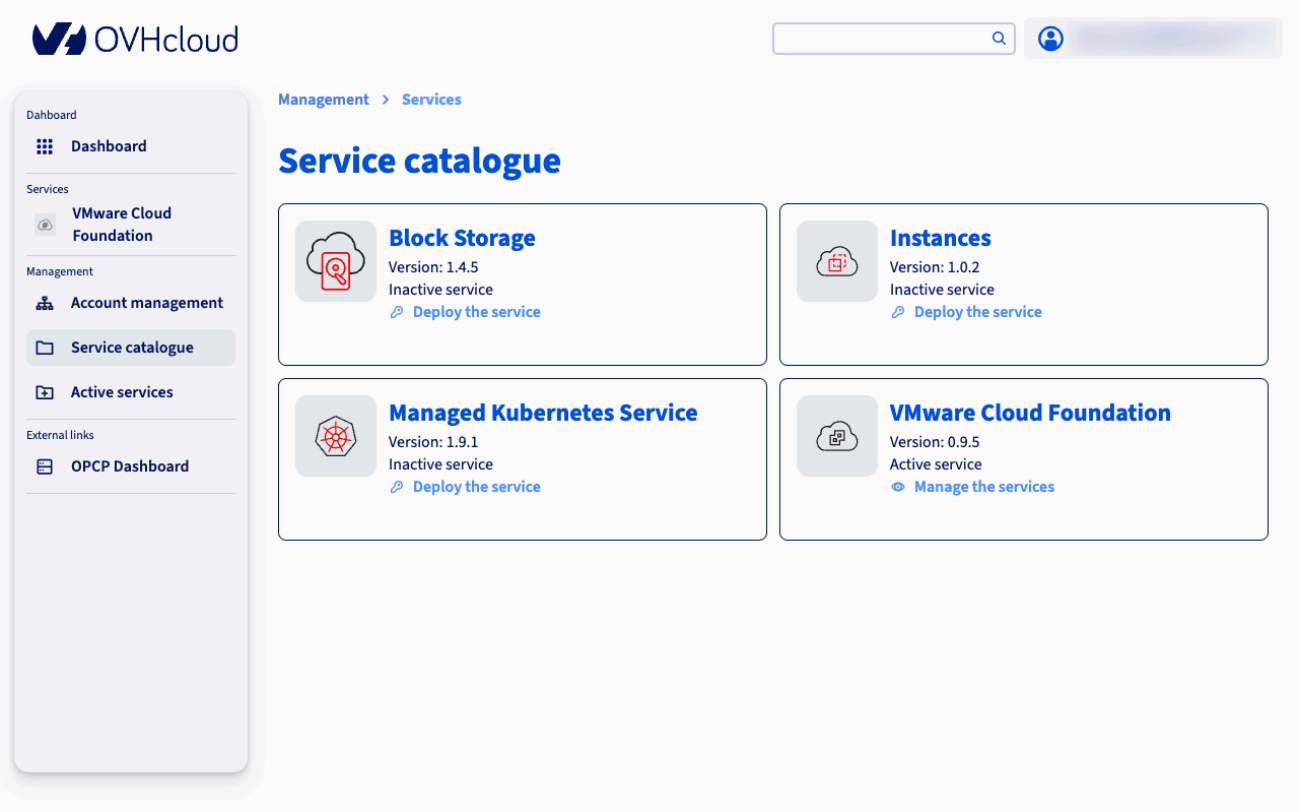
- Create a dedicated Keycloak realm named `account-{name}`.
- Create an admin user with the `account-admin` role.
- Set up a temporary password (the account admin will be prompted to change it on first login).
- Configure a `landing-zone` client for Landing Zone Manager user access.

Deploying a service

Deploying a service is a two-step process: first deploy the **controller**, then deploy one or more **apps** for specific accounts.

Step 1 — Deploy a controller

1. Navigate to the **Service catalog** from the dashboard.



2. Select the service you want to activate.
3. Click **Activate the service**.
4. Choose the version and configure the required properties.
5. Select the hosts on which the controller will be deployed.
6. Submit the form.

> **Info**
>
> Controller deployment is **asynchronous**. The interface will confirm that the deployment has started, but provisioning happens in the background through Kubernetes and Terraform. You can monitor the deployment status from the controllers page.

Step 2 — Deploy an app

Once the controller is active and at least one account exists:

1. Navigate to the controller page.
2. Click **Deploy an app**.
3. Select the target account.
4. Choose the version and configure the required properties.
5. Select the hosts for the app deployment.
6. Submit the form.

During app deployment, the platform automatically creates the necessary Keycloak resources (client-id, roles, and groups) in the target account's realm, ensuring that only authorised users of that account can access the app.

> **Info**
>
> Like controller deployment, app deployment is **asynchronous**. The actual infrastructure provisioning is handled by Kubernetes and Terraform in the background.

Scaling capacity

You can add or remove hosts from a deployed controller or app to adjust capacity.

1. Navigate to the controller or app page.
2. Click **Expand capacity**.
3. Select the hosts to add.
4. Submit the form.

The scaling operation is asynchronous. Kubernetes reconciles the configuration and provisions resources on the updated hosts.

Authentication and access levels

CloudStore uses a layered Keycloak federation model that mirrors the platform architecture:

Level	Keycloak instance	Users	Purpose
L1	OPCP Core Keycloak	DC operators, Super admins	Infrastructure-level identity
L2	CloudStore Keycloak	IT admins, Service admins	Platform management
L3	Per-account Keycloak realms	Landing Zone Manager users	Application access

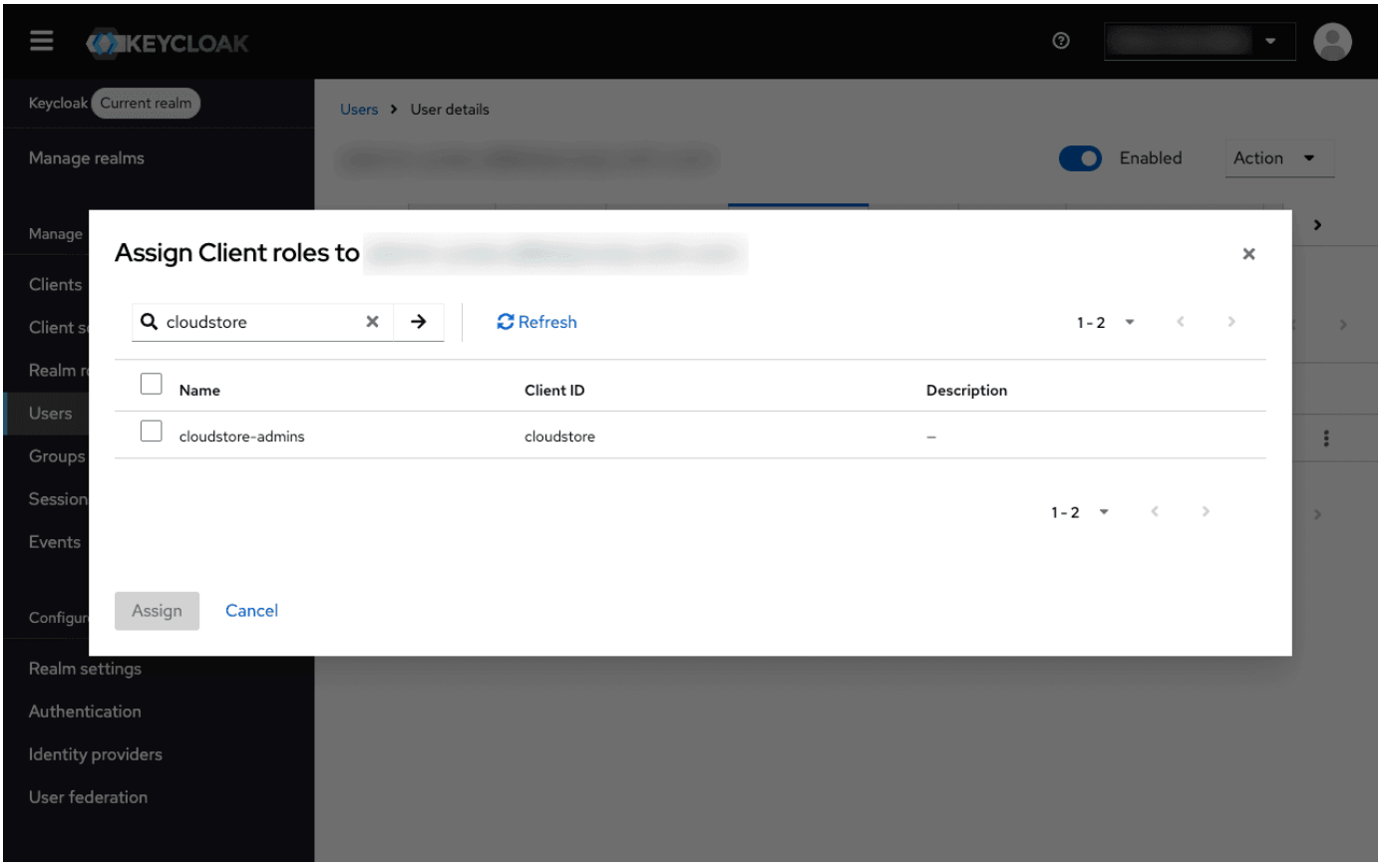
- **Keycloak L2** is federated with **L1** (OPCP Core). This means rights granted on OpenStack projects at L1 are carried through to L2.
- **Keycloak L3** is an independent instance managed by the CloudStore API. Each account gets its own isolated realm.

> **Info**
>
> The **L3 layer** is provided by the Landing Zone Manager, a separate OPCP product responsible for managing Landing Zone Manager user accounts. Its Keycloak stack is **not federated** with the L1 (OPCP Core) and L2 (CloudStore) Keycloak instances.

Managing IAM on CloudStore Keycloak (L2)

To be able to manage users, roles, and groups on the CloudStore Keycloak (L2), first assign the `cloudstore-admins` role to your user in the **OPCP Core Keycloak (L1)**.

1. Log in to the OPCP Core Keycloak administration console (L1).
2. Navigate to the user you want to grant IAM management rights to.
3. In the **Role mappings** tab, assign the `cloudstore-admins` client role.



Once this role is assigned, the user will have the necessary permissions to administer the CloudStore Keycloak (L2), including managing realms, clients, users, and roles.

The Landing Zone

The **Landing Zone** is the interface for Landing Zone Manager users (cloud users). It provides a simplified view of the apps deployed for their account.

Landing Zone Manager users authenticate through their account-specific Keycloak realm (L3) and can only access apps deployed for their account. The Landing Zone retrieves the list of accessible apps and filters them based on user permissions.

Go further

If you need training or technical assistance for the implementation of our solutions, contact your sales representative or click this link to request a quote and have your project analyzed by our Professional Services team experts.

Join our community of users.

Landing Zone Manager

Landing Zone Manager - Creating a user account

Find out how to create a user account in the Landing Zone Manager and log in for the first time

Objective

The Landing Zone Manager is the OPCP portal that lets each team, department or customer deploy and operate their workloads autonomously, in a dedicated and isolated space. Administrators provision users and grant them access to the platform. Each user account is backed by a dedicated realm in the underlying Keycloak instance, ensuring multitenant access isolation.

This guide explains how to create a user account in the Landing Zone Manager and log in for the first time.

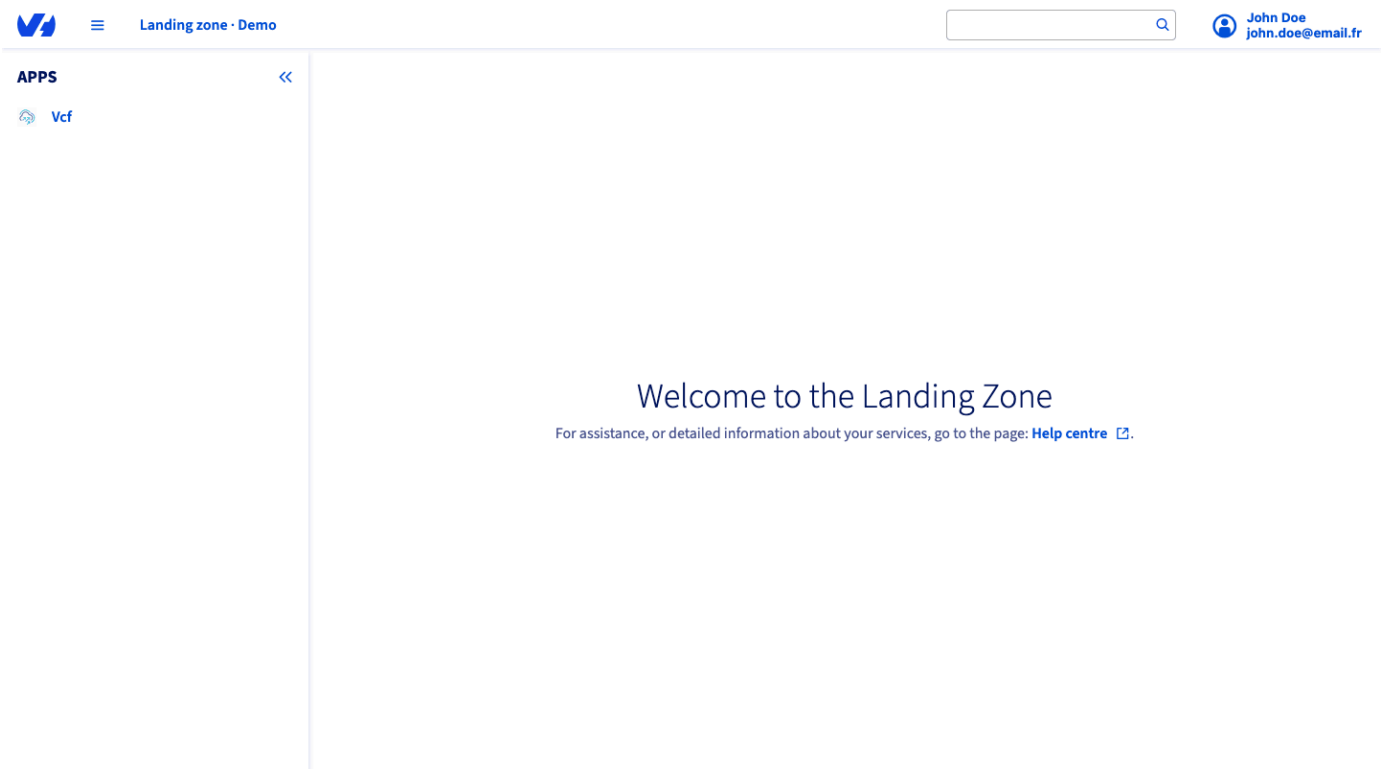
Requirements

- Access to the Landing Zone Manager with administrator privileges allowing user account management
- The first-login default password defined during your Landing Zone Manager deployment
- The URL of the Landing Zone Manager (previously communicated by your administrator)
- Valid user information to provision (full name and email address)

Instructions

Step 1: Access the user account management section

Log in to the Landing Zone Manager with an account that has administrator privileges. From the main navigation, open the **Account management** section.



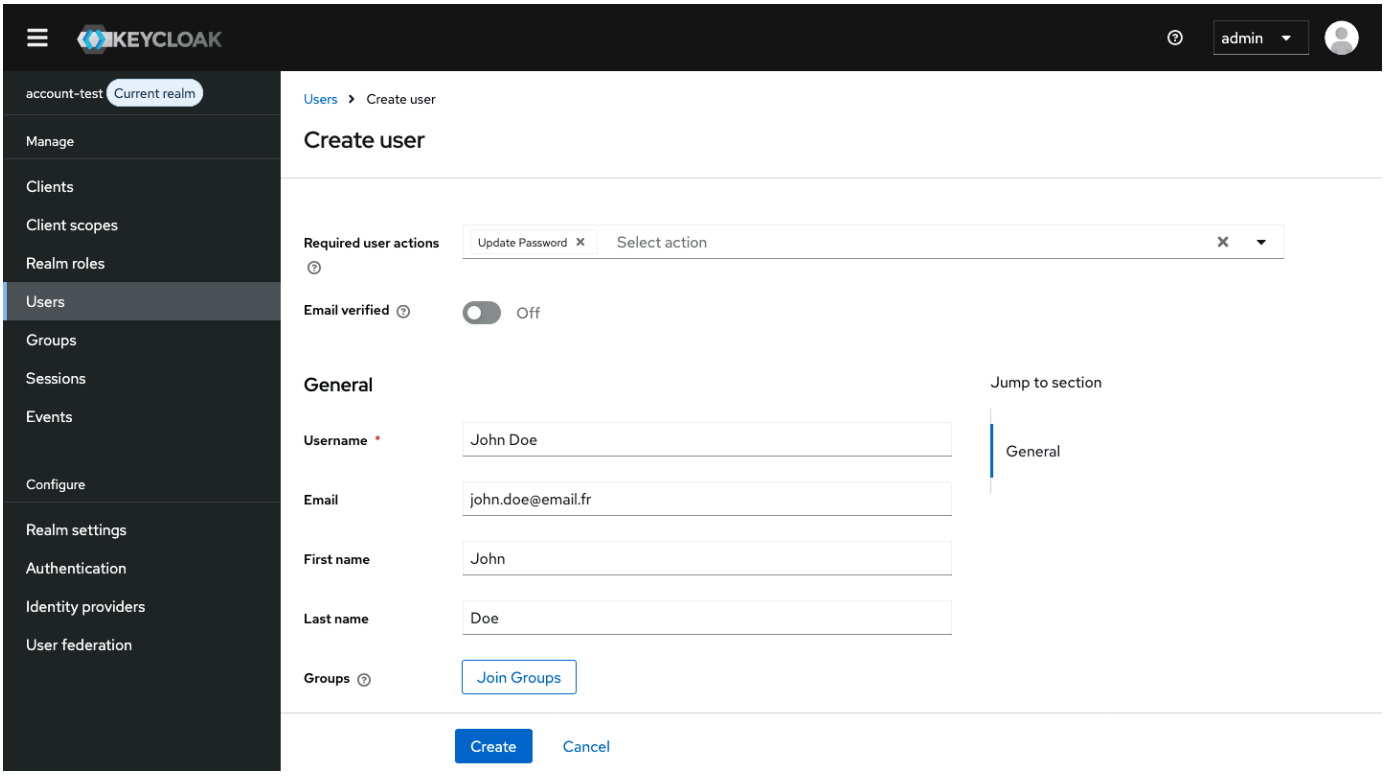
Step 2: Create a new user account

Click the **+ Create new account** button to open the user account creation form.

Fill in the required fields:

Field	Description
Name	The display name of the user account
Email	The user's email address. This value will also be used as the **login** identifier
First name	The user's first name
Last name	The user's last name

Once all fields are filled in, confirm the creation of the user account.



> **Info**
>
> The email address provided is used as the login identifier for the new user account. Make sure it is correct before validating, as users will authenticate with this value.

Step 3: Share the first-login credentials

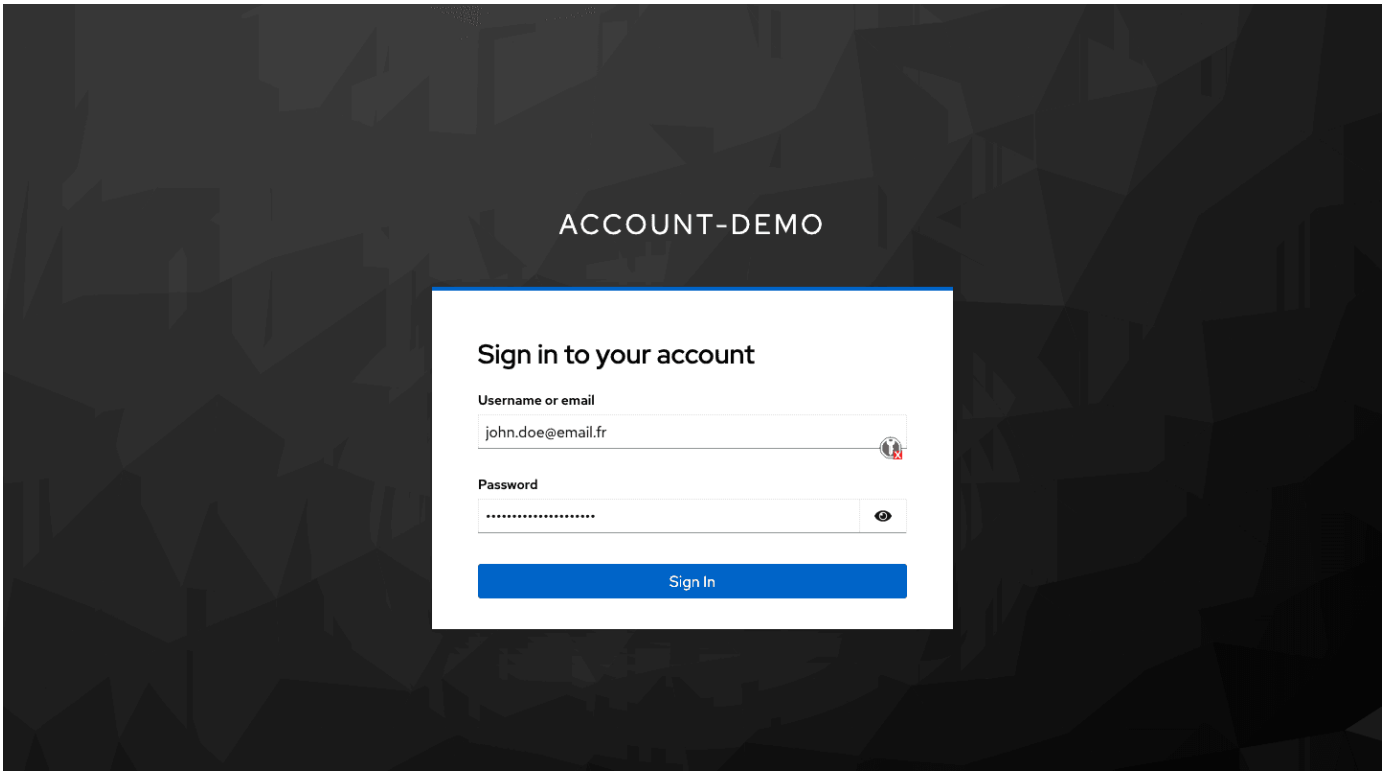
Once the user account is successfully created, share the Landing Zone Manager URL with the user, along with their first-login credentials:

- **Login**: the email address entered during creation
- **Password**: the first-login default password defined during OPCP and CloudStore deployment

> **Warning**
>
> The default password is shared across first logins and is defined at deployment time. For security reasons, users must change this password immediately after their first successful login.

Step 4: First connection to the Landing Zone Manager

Once the user account has been provisioned, the user can connect to the Landing Zone Manager using the URL previously communicated.



From the login page:

1. Enter the email address used when creating the account as the login.
2. Enter the first-login default password.
3. Validate to access the Landing Zone Manager.

On this first login, the user must set a new password and complete any additional authentication setup required by the platform before accessing the Landing Zone Manager.

How user accounts are mapped in Keycloak

> **Info**
>
> The Landing Zone Manager runs on its **own dedicated Keycloak stack**, independent from the OPCP Core Keycloak and from any CloudStore Keycloak. It is not federated with these instances: identities, realms and credentials managed in the Landing Zone Manager are fully isolated from administrators who can access the rest of the OPCP identity layers.

Every account created in the Landing Zone Manager automatically generates a dedicated **realm in Keycloak**. This design ensures:

- independent configuration of authentication flows per account
- separate user bases and role mappings per realm

> **Info**
>
> Because each account corresponds to a distinct Keycloak realm, any identity-related operation (adding users, configuring federation, defining roles) must be performed within the realm associated with the target account.

Go further

For training or technical assistance implementing our solutions, contact your sales representative or visit our Professional Services page to request a quote and have your project analyzed by our experts.

Join our community of users.

Additional resources

OPCP Core compatibility matrix

Find out which OS images and platforms are validated for OPCP Core bare metal provisioning — Ironic features, hardware qualification, and available documentation

Objective

OPCP Core provides bare metal provisioning through OpenStack Ironic. Because this requires specific hardware support and validated firmware versions, not every OS image works out of the box — and firmware updates can break compatibility.

This guide is a **public compatibility matrix** listing platforms that have been fully qualified on OPCP Core. For each, it documents:

- Whether the platform is **available upon delivery** (provisioned directly by OVHcloud without a custom image).
- Which **Ironic features** are supported: software RAID, Active/Backup bonding, LACP (802.3ad).
- Whether **Neutron trunking** is available post-install.
- Links to the relevant **image build instructions**.

Compatibility matrix

The table below lists platforms with status **Ready** — those that have been fully qualified on OPCP Core hardware.

Built-in images

Platform	Ironic software RAID	Ironic Active/Backup bonding	Ironic LACP (802.3ad)	Neutron trunking
Debian 12	Yes	Yes	Yes	Yes
Debian 13	Yes	Yes	Yes	Yes
CentOS Stream 9	Yes	Yes	Yes	Yes

Image build instructions

Platform	Image build guide	Ironic software RAID	Ironic Active/Backup bonding	Ironic LACP (802.3ad)	Neutron trunking
Debian 12	Guide	No	Yes	Yes	Yes
Debian 13	Guide	No	Yes	Yes	Yes

Feature explanations

Ironic software RAID

Software RAID is configured by Ironic during bare metal provisioning. When enabled, the OS is deployed across multiple disks (typically RAID-1 for two-disk configurations), providing local storage redundancy without requiring a hardware RAID controller.

> **Warning**
> Software RAID with Ironic requires GRUB to be built with `mdadm` modules. This must be taken into account when building a custom OS image.

Ironic Active/Backup bonding

Ironic can configure Active/Backup network bonding during provisioning. This provides link redundancy: one NIC is active while the other stands by, and takes over automatically if the primary fails. No configuration is required on the upstream switch.

Ironic LACP (802.3ad)

LACP bonding (IEEE 802.3ad) aggregates two network interfaces into a single logical link, providing both redundancy and increased throughput. This mode requires the upstream switch to support and be configured for LACP.

For step-by-step instructions, see How to setup LACP on a Node.

Neutron trunking

Neutron trunking allows a bare metal instance to carry multiple Neutron networks over a single physical port using VLAN tagging. It is configured post-installation through OpenStack Neutron, and not during the Ironic provisioning phase.

> **Info**
> OpenStack does not configure VLAN trunking natively through cloud-init. Enabling trunking requires post-install configuration steps detailed in How to setup trunk on a Node.

Go further

- How to setup an instance using the OpenStack CLI
- How to setup LACP on a Node
- How to setup trunk on a Node
- How to setup Softraid on a Node

For training or technical assistance implementing our solutions, contact your sales representative or visit our Professional Services page to request a quote and have your project analyzed by our experts.

Join our community of users.

Object Storage features and specifications on OPCP

Discover the specifications of Object Storage for OVHcloud OPCP, with information on bucket access, performance classes, encryption and versioning.

> **Info**
>
> Please note that this information may change prior to General Availability (GA). Our current estimate for GA is **March 2026**. We will make every effort to keep you informed of any updates, but we recommend checking this guide regularly for the latest information. Thank you for your understanding.

Objective

OVHcloud Object Storage in **OPCP** provides a scalable, durable, and low-latency solution for storing and accessing unstructured data. Designed to meet the needs of various applications, it offers a robust platform for data storage with OPCP-specific optimizations to ensure high performance and availability.

How is bucket access managed?

All access keys of a project can access all buckets in OPCP.

Performance classes

- **Standard**: the performance classes in OPCP are the same as the standard performance of OVHcloud, ensuring consistency in data processing and access speeds.
- **High Performance**: High Performance storage classes are not available in OPCP at the moment. Only Standard classes are supported.

Support for High Performance classes in OPCP may be considered in the future by OVHcloud.

Features

The features of Object Storage in OPCP are mainly the same as in OVHcloud regions. Some important differences are detailed in the comparison table in this guide.

MultiAZ and MonoAZ specifications

MultiAZ (Multi-availability zone)

- **Overview**: MultiAZ ensures high availability and redundancy by distributing data across multiple availability zones within the same region, guaranteeing data access even in the event of a zone failure.
- **Availability**: MultiAZ configurations are available in regions but are not supported in OPCP, which focuses on low-latency access within a single zone.

MonoAZ (Single-availability zone)

- **Overview**: MonoAZ stores data in a single availability zone, offering **reduced latency and high performance** for applications that do not require MultiAZ redundancy.
- **Availability**: MonoAZ is supported in OPCP, suitable for use cases prioritizing performance and low latency.

Metadata API (unavailable)

- **Overview**: The metadata API, which allows retrieving storage information, is not available in OPCP. This includes the number of buckets and total size via the customer area or API.
- **Impact**: Users cannot track or manage their storage usage programmatically in OPCP.

Our teams are working on the future integration of this API.

Encryption and versioning

Encryption

- **Current status**: Encryption at rest is not supported in OPCP.
- **Supported encryption**: Client-side encryption (SSeC) is available, allowing users to encrypt their data before sending it to the Object Storage service.

Versioning

- **Current status** : versioning is supported by the API, allowing the management of multiple versions of an object in a bucket.
- A more comprehensive versioning integration will be proposed in future API updates.

User policies (unavailable)

- **Overview**: *User policies* can theoretically control access, but they do not affect the OPCP model.
- **Impact**: All access keys of a project have unlimited access to all OPCP buckets, which may not meet certain security requirements.

Improvements for user policy management are planned for future updates.

Go further

If you need training or technical assistance for the implementation of our solutions, contact your sales representative or click this link to request a quote and have your project analyzed by our Professional Services team experts.

Join our community of users.

Building a custom Debian image for OPCP

Find out how to create a custom Debian image for On-Prem Cloud Platform

Objective

This guide focuses on building customized Debian disk images for both OpenStack Ironic (bare metal) and Nova (compute) using diskimage-builder.

It's a good starting point to understand how diskimage-builder (DiB) works with a practical example. We will build a custom Debian 13 image from the upstream image and customize it with Ansible.

At the end, we will generate a `debian13.qcow2` whole disk image (with the kernel and initramfs), ready to be imported into OpenStack Glance.

OpenStack Ironic/Nova expectations

Image Format

- **qcow2** (recommended): Compressed and efficient for Glance storage
- **raw**: Uncompressed disk image

The image should be a **whole disk image** that includes:

- Boot sector/EFI system partition
- Operating system partition(s)
- Kernel and initramfs embedded in the disk

Partitioning and Boot Requirements

For BIOS Boot:

- GPT or MBR partition table
- BIOS boot partition (1-2MB, type `ef02` for GPT)
- Root partition with bootloader installed (GRUB2)
- Bootloader must be installed to MBR/boot sector

For EFI Boot:

- GPT partition table required
- EFI System Partition (ESP): 512MB, FAT32, mounted at `/boot/efi`
- Root partition with GRUB2 EFI bootloader
- EFI boot entries properly configured

For Ironic Bare Metal (Recommended Configuration):

Standard Partition Layout (Simple):

```
# GPT partition table
- EFI System Partition (ESP): 512MiB, type EF00, FAT32, mounted at /boot/efi
- BIOS Boot Partition (BSP): 8MiB, type EF02 (for hybrid BIOS/EFI compatibility)
- Root partition: remaining space, type 8300, ext4, mounted at /
```

Requirements

System Requirements:

- Root permissions
- At least 10GB available
- Some packages:

```
# Debian/Ubuntu
apt update -y && apt install -y \
  dosfstools \
  python3 \
  python3-venv \
  python3-pip \
  virtualenv \
  kpartx \
  debootstrap \
  lvm2 \
  squashfs-tools \
  qemu-utils
```

Install diskimage-builder (DiB) in a virtualenv:

```
mkdir -p diskimage-builder
python3 -m venv venv
. ./venv/bin/activate
pip install diskimage-builder
```

Understanding diskimage-builder Elements

Elements are modular components that customize your image.

Directory	Purpose
`environment.d/`	Environment variables
`extra-data.d/`	Files added before installation
`pre-install.d/`	Pre-installation scripts
`post-install.d/`	Post-installation scripts
`finalise.d/`	Final customization
`cleanup.d/`	Cleanup tasks
`package-installs.yaml`	Package declarations
`block-device-default.yaml`	Disk partitioning

Scripts in these directories execute in numerical/alphabetical order.

Creating a Customization Element with Ansible

Create an element that uses Ansible for customization:

Directory Structure:

```
mkdir -p elements/os-custom/extra-data.d/ansible/{roles/customize/tasks,inventory/host_vars/sys_image}
mkdir -p elements/os-custom/extra-data.d/ansible/roles/customize/files
mkdir -p elements/os-custom/post-install.d
```

Post-install Scripts:

`elements/os-custom/post-install.d/00-install-ansible`:

```
#!/bin/bash
set -eux
apt install --yes ansible
```

`elements/os-custom/post-install.d/01-apply-ansible`:

```
#!/bin/bash
set -eux
export LANG=C.UTF-8
export LC_ALL=C.UTF-8
cd /tmp/in_target.d/extra-data.d/ansible
ANSIBLE_STDOUT_CALLBACK=debug ansible-playbook -i inventory main.yml
```

`elements/os-custom/post-install.d/02-remove-ansible`:

```
#!/bin/bash
set -eux
apt remove --yes ansible
apt autoremove --yes
```

Make scripts executable:

```
chmod +x elements/os-custom/post-install.d/*
```

Ansible Configuration:

`elements/os-custom/extra-data.d/ansible/main.yml`:

```
---
- hosts: all
  gather_facts: true
  roles:
    - name: customize
```

`elements/os-custom/extra-data.d/ansible/inventory/hosts`:

```
[all]
sys_image ansible_connection=local ansible_become=no
```

In this example, we are using `cloud-init` with the `netplan` renderer to configure `systemd-networkd`. This is a working example of customizing network configuration; feel free to adapt it to your needs.

Note: When Ironic configures bare metal on first boot, it will propagate a `network_metadata` manifest (configdrive) that can be interpreted by `cloud-init` to automatically configure the network (such as static IP, LACP, etc.).

`elements/os-custom/extra-data.d/ansible/roles/customize/tasks/main.yml`:

```
---
- name: Install additional packages
  apt:
    name:
      - cloud-init
    state: latest
    update_cache: yes

- name: Remove unwanted packages
  apt:
    name:
      - ifupdown
      - ifenslave
      - vlan
    state: absent
    purge: yes

# Allow Baremetal LACP auto-conf from Neutron
- name: Configure cloud-init for netplan
  copy:
    src: 50-netplan.cfg
    dest: /etc/cloud/cloud.cfg.d/50-netplan.cfg
    owner: root
    group: root
    mode: 0644
```

`elements/os-custom/extra-data.d/ansible/roles/customize/files/50-netplan.cfg`:

```
system_info:
  network:
    renderers: ['netplan']
```

Additional Packages:

`elements/os-custom/package-installs.yaml`:

```
man:
nano:
tcpdump:
iputils-ping:
netplan.io:
```

Element Dependencies:

`elements/os-custom/element-deps`:

```
debian
```

Building the Image

Create an environment file `debian13.env`:

```
export DIB_RELEASE=trixie
export DIB_CLOUD_INIT_DATASOURCES="ConfigDrive, OpenStack"
export DIB_GRUB_TIMEOUT=10
```

Build the image:

```
mkdir -p tmp-build-dir
export TMPDIR="$(pwd)/tmp-build-dir"
export ELEMENTS_PATH="$(pwd)/elements"

source debian13.env
disk-image-create -t qcow2 --image-size 16GB -a amd64 vm block-device-efi os-custom debian -o debian13
```

You should get a file `debian13.qcow2`.

If you want environment and packages SBOM files:

```
cp debian13.d/dib-manifests/dib_environment debian13.env.sbom
cp debian13.d/dib-manifests/dib-manifest-dpkg-debian13 debian13.pkg.sbom
```

Testing the Image

A quick way to test the generated image is using qemu to spawn a virtual machine using the qcow2 image and a VNC client to connect to the monitor. We can test both EFI and BIOS boot.

BIOS Boot

```
qemu-system-x86_64 -enable-kvm -vnc 0.0.0.0:0,password=on -monitor stdio -m 2048 -drive
file=debian13.qcow2,if=virtio,format=qcow2
```

```
# Set up a custom VNC password
QEMU 10.0.3 monitor - type 'help' for more information
(qemu) change vnc password
Password: ********
```

Use any VNC client to connect :5200

EFI

```
# Copy OVMF vars to avoid modifying the original
cp /usr/share/OVMF/OVMF_VARS_4M.fd /tmp/debian13-OVMF_VARS_4M.fd

qemu-system-x86_64 -enable-kvm \
  -machine q35,smm=on,accel=kvm \
  -drive if=pflash,format=raw,unit=0,file=/usr/share/OVMF/OVMF_CODE_4M.fd,readonly=on \
  -drive if=pflash,format=raw,unit=1,file=/tmp/debian13-OVMF_VARS_4M.fd \
  -vnc 0.0.0.0:0,password=on \
  -monitor stdio \
  -m 2048 \
  -drive file=debian13.qcow2,if=virtio,format=qcow2
```

```
# Set up a custom VNC password
QEMU 10.0.3 monitor - type 'help' for more information
(qemu) change vnc password
Password: ********
```

Use any VNC client to connect :5200

Upload Image to OpenStack:

```
openstack image create \
  --disk-format qcow2 \
  --container-format bare \
  --file debian13.qcow2 \
  debian13
```

Done! You can now create a baremetal instance or compute instance with the new created image.

Ceph RBD Block Storage - Performance, Resilience and Scalability with OpenStack

Discover OVHcloud OPCP's Ceph RBD Block Storage: performance, high availability and advanced features for your OpenStack environments

> **Info**
>
> Please note that this information may change prior to General Availability (GA). Our current estimate for GA is **March 2026**. We will make every effort to keep you informed of any updates, but we recommend checking this guide regularly for the latest information. Thank you for your understanding.

Objective

This guide presents the Block Storage architecture implemented in **OPCP**, based on OpenStack. It explains how persistent storage for virtual machines (VMs) is ensured by **Ceph RADOS Block Device** (RBD), an open-source, distributed and highly scalable solution designed to meet the needs of modern enterprises.

The objective is to show how this storage layer guarantees performance, resilience and scalability, allowing teams to focus on application development and deployment without being limited by the infrastructure.

1. Performance and Scalability: Designed to evolve

Unlike traditional storage solutions, the Ceph infrastructure distributes data and workloads across all physical nodes in the cluster.

This approach eliminates bottlenecks and allows for linear scalability suitable for demanding environments.

Feature	Customer Benefit
Distributed I/O	**Elimination of bottlenecks:** read and write operations are simultaneously distributed across many devices. This distribution ensures **high throughput** and **low and constant latency**, essential for critical applications and databases.
Horizontal scalability	**Growth without interruption:** increasing storage capacity is simply achieved by adding standard servers to the cluster. The system automatically integrates the new hardware and rebalances the data in the background, **without causing any downtime**.
Dynamic provisioning	**Optimized efficiency:** volumes are allocated instantly but only use physical space when data is actually written. This approach significantly improves the **utilization and efficiency** of the physical storage pool.

2. Resilience and data protection

The fundamental architecture of Ceph is designed to offer **maximum fault tolerance**, ensuring that hardware failures do not cause service interruptions or data loss.

Automatic replication (High availability):

- Each piece of data is automatically copied and stored on multiple distinct disks and physical servers (typically 3 copies).
- In the event of a disk or server failure, virtual machines continue to operate normally, accessing the remaining copies without interruption.

Self-healing:

- When a device fails, Ceph immediately detects the anomaly and replicates the lost data onto the remaining healthy devices.
- This process automatically restores the **defined data protection level**.

No single point of failure:

- The system operates without a central controller or proprietary hardware component, ensuring **real redundancy** and **maximum availability** for cloud services.

3. Technical specifications and limitations

This section details the main technical features and compatibility requirements for users deploying applications on the OVHcloud OPCP platform.

A. Compatibility and access

Element	Detail	Description
Storage type	Block storage	Presented to the VM as a standard SCSI or virtio block device, equivalent to a physical disk for optimal performance.
POSIX compatibility	Yes (via guest OS)	Ceph RBD provides the raw block device; the guest OS file system (XFS, ext4, NTFS, etc.) ensures POSIX compliance.
Network file system	No	It is not a shared NFS or SMB. Each volume is dedicated to a single VM at a time for maximum performance.

B. Limitations and sizing

Metric	Value/Default	Notes
Maximum volume size	16 TB	Limit to ensure optimal performance and ease of management under OpenStack. Custom pools can support larger sizes on request.
Minimum volume size	1 GB	Smallest deployable volume unit via Cinder.
Volume availability	Instant	Volume allocation and attachment are done immediately via the OpenStack API or dashboard.

C. Performance and quality of service (QoS)

The platform offers several **performance levels** via Cinder volume types (Storage Classes), with guaranteed I/O limits enforced by the Ceph RBD QoS system:

Storage class	IOPS	Throughput	Use case
Distributed storage (Standard)	1 000 / 1 000	100 MB/s	General VMs, web servers, test and development environments.
Distributed storage (High)	3 000 / 3 000	200 MB/s	Databases, analytics engines, high-traffic or latency-sensitive applications.
Distributed storage (Insane)	7 500 / 7 500	300 MB/s	Extreme workloads, critical transactional systems, deep learning infrastructure.

> **Info**
>
> **Note:** these thresholds are enforced by the Ceph QoS system. Volumes may sometimes exceed these limits if system resources allow, but the guarantees ensure predictable performance, even during high demand periods.

4. Advanced data management features

The **Ceph RBD** backend offers advanced tools for volume lifecycle management, simplifying and accelerating operations for cloud users.

- **Instant snapshots:** Creation of **space-efficient point-in-time copies** of any volume, allowing for quick rollback or the creation of new environments in seconds.
- **Fast cloning (Copy-on-Write):** Rapid provisioning of new **read/write volumes** from snapshots or existing images. The **CoW (Copy-on-Write)** technology saves disk space and accelerates VM deployment.
- **Hot volume resizing:** Increasing the size of a volume attached to a running VM, **without shutting down or restarting the VM.**
- **Support for live migration:** The distributed storage allows for non-disruptive migration of active virtual machines between compute nodes.
- **Boot from volume:** Direct launch of a VM from a Cinder volume, ensuring that the root disk immediately benefits from **all the high availability and advanced features of Ceph RBD.**
- **Disaster recovery ready (RBD Mirroring):** The underlying Ceph architecture supports **asynchronous volume mirroring** to a remote Ceph cluster, offering a robust solution for **Disaster Recovery Planning (DRP)**.

Go further

If you need training or technical assistance for the implementation of our solutions, contact your sales representative or click this link to request a quote and have your project analyzed by our Professional Services team experts.

Join our community of users.