IONOS CLOUD

Red Hat OpenShift Container Platform 4.17 on IONOS CLOUD: Deployment Guide

Version 1.4, May 15th, 2026

Table of Contents

1 Introduction

1.1 Purpose of the document

Regardless if a customer's application landscape is about to be modernized or already consists of applications developed in a microservice architecture, organizations are looking for leading application platforms to build, deploy and operate their most critical business workload.

Many organizations are looking for sovereign and local cloud environments as an alternative to global hyperscalers.

Sovereign cloud usage is often driven by regulatory, security, and operational considerations.

- **Some of the key reasons are:**
    - Data Compliance and Legal Requirements
    - Data Residency and Sovereignty
    - Security and Risk Mitigation
    - Independence from Foreign Cloud Providers
    - Digital Sovereignty and Strategic Autonomy
    - Economic and Innovation Benefits
    - Trust and Transparency

As a leading sovereign cloud provider, IONOS CLOUD is partnering with Red Hat to enable Red Hat OpenShift Container Platform deployment.
The goal of this document is to describe a semi-automated deployment of OpenShift Container Platform on top of IONOS CLOUD.

This comprehensive walkthrough for different installation scenarios will help to keep aligned with recommended practices for initial installation and also provides valuable insights for typical operational tasks in the “Day 2” section of the document.

Within this deployment guide, a default OpenShift Container Platform High Availability Cluster (consisting of 3 Control-Plane Nodes and 2 Worker Nodes) will be described, as this is the bare minimum recommended for production environments.

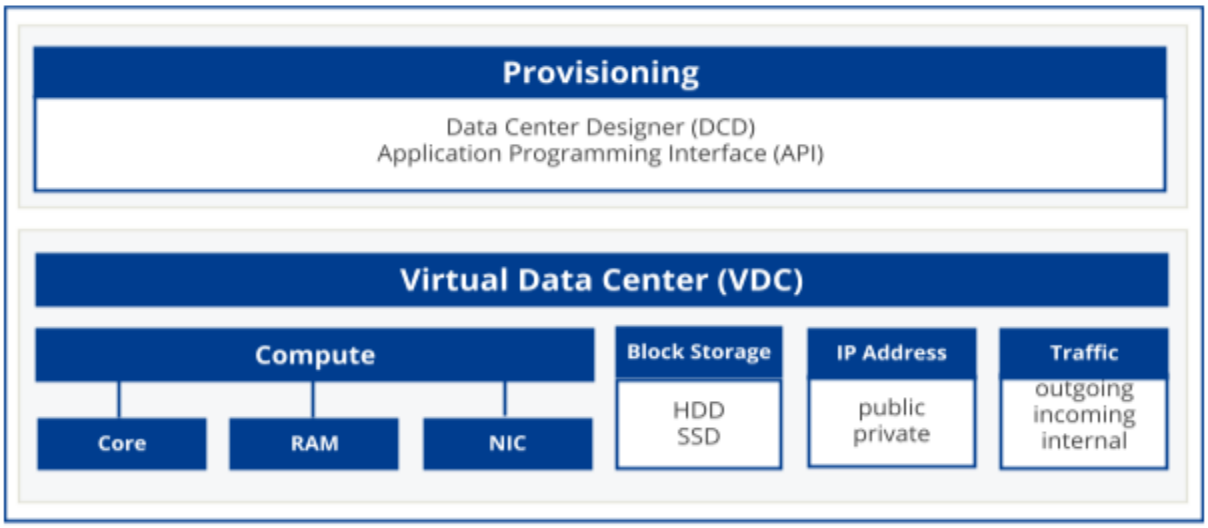
Please note that the deployment guide is partially focused on manual steps to deploy the OpenShift Container Platform Cluster on IONOS CLOUD and does not describe a fully automated installation approach.
Of course, further automation is possible and recommended (please check the IONOS CLOUD API documentation for details) but it’s out of the scope of this document.

This deployment guide was also used to validate the OpenShift Container Platform installation on IONOS CLOUD, as referenced in the Red Hat Ecosystem catalog.

1.2 Introduction to IONOS CLOUD

IONOS CLOUD offers its customers "Infrastructure as a Service" (IaaS) in the form of virtual computing, data storage, and network resources. The customer is able to make use of these resources on a flexible basis as required. The resources used (Cores/vCPUs, RAM, Storage) are billed to the customer by the minute based on a price list, which is valid at the time. Billing of external data transfers is based on data volume.

1.2.1 IONOS CLOUD Compute Engine model



1.2.2 Data Center Designer

IONOS CLOUD provides the customer with access to a personalized web application called the “Data Center Designer” (DCD). The DCD can be accessed via modern Internet browsers.

1.2.3 Cloud API

IONOS CLOUD provides the customer with an Application Programming Interface (API). This API gives the customer automated control over the functions from the DCD. Upon request, IONOS CLOUD will provide an API reference along with example software (Cloud-CLI) on how the Cloud API can be used (links below).

IONOS CLOUD provides access to the Cloud functionality for developers based on REST (Representational State Transfer). All account types are able to use the Cloud API.

Scope	URL
Cloud API Documentation	https://api.ionos.com/docs/cloud/v6/
Cloud API Endpoint	https://api.ionos.com/cloudapi/v6/

1.2.4 Virtual Data Center

On the IONOS CLOUD platform, the customer can create so-called “Virtual Data Centers” (VDC). A VDC is a repository for all infrastructure resources ordered by the customer. Access to the resources in a VDC – similar to operating a physical data center – is only possible via a corresponding network or internet connection. Within a VDC, the IONOS CLOUD software allows for the distribution of various resources to different availability zones.

For more information regarding the IONOS CLOUD, please refer to the IONOS CLOUD Service Catalog.

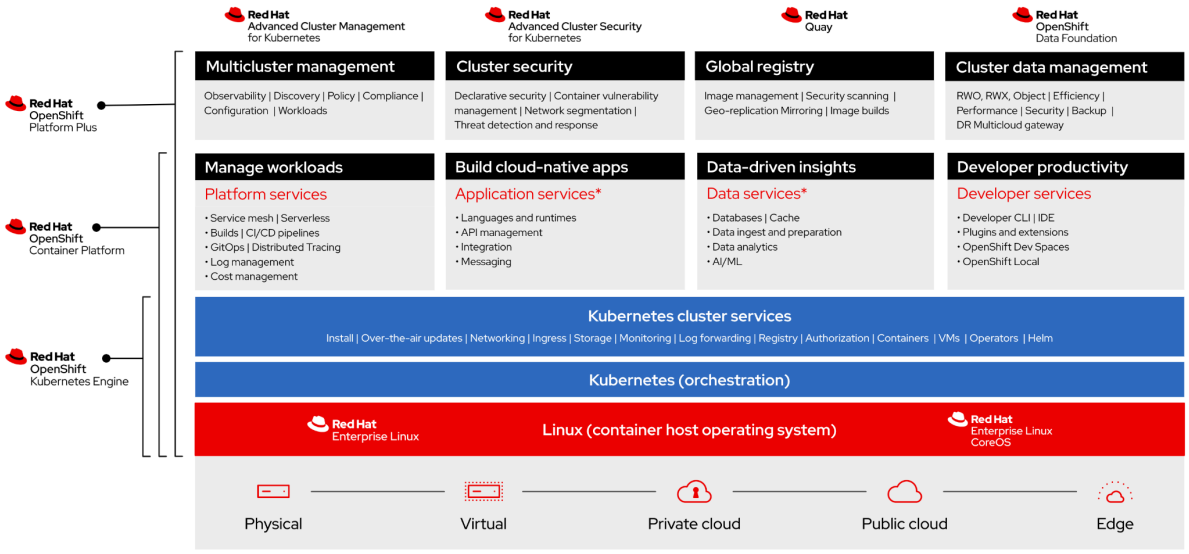
1.3 Introduction to Red Hat OpenShift Container Platform

Red Hat OpenShift Container Platform is a trusted, comprehensive, and consistent platform to develop, modernize, and deploy applications at scale, including today’s AI-enabled apps. It enables businesses to innovate faster with a complete set of services for bringing apps to market on your choice of infrastructure.

OpenShift Container Platform is the leading hybrid cloud application platform, bringing together a comprehensive set of tools and services that streamline the entire application lifecycle, from development to delivery to management of app workloads.

Trusted by 3,000 customers across industries (including 56% of the top 25 Global Fortune 500), it combines built-in security features with dedicated support, a trusted software supply chain, and Red Hat Enterprise Linux as the operating foundation.

OpenShift Container Platform offers a complete and consistent application platform that can be fully managed in the public cloud or self-managed in any environment, offering a more integrated and streamlined platform for innovation while reducing operational complexity. The different functionalities and layers are visualized in the following graphics:



The architecture of Red Hat OpenShift Container Platform consists of the following key components:

- **Control Plane** (Master Nodes)
  - API Server – Manages Kubernetes API requests.
  - Controller Manager & Scheduler – Assigns workloads and maintains cluster state.
  - etcd – Stores cluster metadata.
  - OAuth Server – Handles authentication & authorization.

- **Worker Nodes** (Compute Nodes)
  - Kubelet & CRI-O – Runs containers.
  - Kube Proxy & SDN – Manages networking and inter-pod communication.

- **OpenShift-Specific Enhancements**
  - Router (HAProxy) – Ingress traffic management.
  - Operators & OLM – Automates app deployment.
  - Persistent Storage – Supports PVs & CSI-based storage.
  - Monitoring & Logging – Uses Prometheus, LokiStack and OpenTelemetry.

- **CI/CD & Developer Tools**
  - OpenShift Pipelines (Tekton) – Cloud-native CI/CD automation.
  - Source-to-Image (S2I) – Directly converts source code to container images.
  - GitOps (ArgoCD) – Declarative application deployment.

- **Security & Governance**
  - RBAC & Security Context Constraints (SCCs) – Role-based security policies.
  - Network Policies – Controls pod-to-pod communication.

For more details, please review the Red Hat OpenShift Container Platform documentation.

2 Red Hat OpenShift Container Platform installation details

2.1 Understanding OpenShift Container Platform Installation Method Differences

Red Hat OpenShift Container Platform offers four different installation methods to suit various infrastructure and operational needs:

- **Automated / Installer-Provisioned Infrastructure (IPI)**
  - The OpenShift installer provisions and configures the required infrastructure automatically.
  - Suitable for cloud environments such as AWS, Azure, and vSphere or baremetal host baseboard management controller (BMC) integrations
  - Eases deployment by handling networking, compute, and storage provisioning.

- **Full Control / User-Provisioned Infrastructure (UPI)**
  - Requires users to manually set up and configure the infrastructure before deploying OpenShift.
  - Provides more flexibility and customization for on-premise and bare-metal environments.
  - Best for enterprises with specific networking, security, or hardware requirements.

- **Local Agent-based**
  - Uses pre-configured bootable images (ISO) for streamlined deployments.
  - Best suited for air-gapped or disconnected environments.
  - Reduces manual intervention in large-scale deployments.

- **Interactive (Assisted Installer)**
  - A web-based guided installation provided by Red Hat Hybrid Cloud Console.
  - Simplifies deployment for OpenShift clusters in various environments.
  - Ideal for both connected and disconnected setups, with pre-validation capabilities.

Please review the documentation for more details about the different deployment methods.

Each method supports to deploy a Cluster with the following characteristics:

- Highly available infrastructure with no single points of failure, which is available by default.
- Administrators can control what updates are applied and when.

This deployment guide describes the **Local Agent-based** installation method, as this is the recommended one for setting up Red Hat OpenShift Container Platform on IONOS CLOUD.

Please note that for the validation of Red Hat OpenShift Container Platform on IONOS CLOUD (as referenced in the Red Hat Ecosystem catalog), exactly this installation method was used.

Additionally, we will use some of the infrastructure preparation guidelines from the **Full Control (UPI)** installation method and adapt those to the needs for IONOS CLOUD.

2.2 Understanding connected and disconnected environments

The main difference between connected and disconnected installations of Red Hat OpenShift Container Platform lies in their network connectivity and access to external resources:

- **Connected Installation**
  - **Internet Access**: Requires direct internet access for all machines in the cluster.
  - **Image Retrieval**: Can pull container images directly from Red Hat's online repositories.
  - **Updates**: Allows for automatic updates and easier access to the latest components.
  - **Operator Hub**: Utilizes the default Operator Hub and external image registries.

- **Disconnected Installation**
  - **Network Isolation**: Operates in a network-restricted environment with limited or no internet access.
  - **Local Resources**: Requires mirroring of all necessary images and content to a local container registry.
  - **Manual Updates**: Needs manual intervention for updates and new component installations.
  - **Custom Configuration**: Uses a local mirror registry to host required images and content, with additional setup steps such as:
    - Configuring custom Network Time Protocol (NTP) settings.
    - Creating a mirror registry for Red Hat OpenShift.
    - Using tools like `oc-mirror` to mirror images locally.
    - Adjusting installation configuration files to point to local resources.

- **Key Considerations for Disconnected Installations**
  - **Security and Compliance**: Provides better control over the environment and reduces external dependencies.
  - **Self-contained Environment**: Ensures all necessary components are available locally without relying on external sources.
  - **Preparation Effort**: Requires more setup steps and resources before installation compared to connected installations.

    - **Update Process**: Involves manually mirroring and applying updates to the cluster.

Disconnected installations are ideal for organizations with strict security requirements or those operating in environments with limited or no internet connectivity. While they offer greater control and isolation, they require more preparation and ongoing manual management than connected installations.

More details are available in the Red Hat OpenShift Container Platform documentation.

This deployment guide describes a connected Red Hat OpenShift Container Platform installation details for IONOS CLOUD.

Please note that for the validation of Red Hat OpenShift Container Platform on IONOS CLOUD (as referenced in the Red Hat Ecosystem catalog), the connected installation was used.

2.3 Understanding OpenShift Container Platform installation topologies

Red Hat OpenShift Container Platform offers several deployment topologies to accommodate various infrastructure needs and use cases:

- **Highly available OpenShift Cluster (HA):**
    - Traditional deployment with multiple nodes
    - Consists of at least three control plane nodes and two or more worker nodes
    - Highly scalable, supporting thousands of instances across hundreds of nodes
    - Provides high availability and fault tolerance
    - Suitable for production environments requiring scalability and complex applications

- **Three-node OpenShift Cluster (Compact):**
    - Reduced minimum system requirements compared to multi-node deployments
    - Consists of three nodes that can run both control plane and applications
    - Balances resource efficiency with high availability
    - Suitable for smaller deployments or environments with limited resources

- **Single-node OpenShift Cluster (SNO):**
    - Allows running OpenShift on a single node
    - Ideal for edge use cases and environments with space or resource constraints
    - Introduced to support edge computing scenarios
    - Offers the core functionality of OpenShift in a minimal footprint

Please note the generally recommended minimum cluster resources for the different topologies and particular check with the requirements for OpenShift on IONOS CLOUD in chapter 3:

Topology	Number of control plane nodes	Number of compute nodes	vCPU	Memory	Storage
Single-node cluster	1	0	8 vCPUs	16 GB of RAM	120 GB
Compact cluster	3	0 or 1	8 vCPUs	16 GB of RAM	120 GB
HA cluster	3 to 5	2 and above	8 vCPUs	16 GB of RAM	120 GB

Please remember: Within this deployment guide, only the default OpenShift High Availability Cluster (consisting of 3 Control-Plane Nodes and 2 Worker Nodes) topology will be described, as this is the bare minimum recommended for production environments **and** required for the Red Hat Ecosystem certification.

Please remember: Minimum requirements, especially for the Control Plane Nodes may not fit to the intended usage of the Red Hat OpenShift Container Platform Cluster. Depending on the expected workload (number of deployments, pods, Operators, …) and the expected size of the Cluster (number of Compute Nodes), it is highly recommended to already start with appropriate resources. More details can be found in the 8.5 Scaling the OpenShift Cluster chapter of the deployment guide.

3 IONOS CLOUD Service and OpenShift Requirements

3.1 Description of relevant IONOS CLOUD services

3.1.1 Compute (IONOS CLOUD Dedicated Core Server)

Under the term “Compute Engine”, IONOS CLOUD offers its customers "Infrastructure as a Service" (IaaS) in the form of virtual computing, data storage, and network resources. The customer is able to make use of these resources on a flexible basis as required. The resources used (Cores/vCPUs, RAM, Storage) are billed to the customer by the minute based on a price list, which is valid at the time.

For **OpenShift Container Platform**, we are using the **Dedicated Core Server** variant:
These virtual machines run on dedicated CPU Cores. With Dedicated Core Servers you gain full access to the provisioned CPU resources,free from resource sharing with other virtual machines on the same physical host. This guarantees optimal performance, stability, reduced latency and predictable performance. You can freely configure the number of cores and RAM required for your workloads, while choosing from the available CPU types available in your current VDC. Dedicated Core Servers can boot from a storage volume, a CD-ROM, or a NIC.

3.1.2 Disk Storage (IONOS CLOUD Block Storage)

IONOS CLOUD Hard Disk Drive (HDD) and Solid State Drive (SSD) Block Storage allow the customer to make use of a dual-redundant storage system. Each block storage created by the customer is stored on two storage servers, providing active-active redundancy. For additional data protection, every storage server is based either on a hardware RAID system or on a software RAID system.

For Solid State Drive volumes, IONOS CLOUD offers two performance classes that can be selected at the time of ordering the volume. SSD Premium is optimized for high performance while SSD Standard is recommended for fast data access with general-purpose scenarios.

For **OpenShift Container Platform**, we are using the **SSD Premium class**.

3.1.3 Object Storage (IONOS CLOUD Object Storage)

IONOS CLOUD Object Storage is a secure, scalable storage solution that offers high data availability and performance.
The product adheres to the S3 API standards, enabling the storage of vast amounts of unstructured data and seamless integration into S3-compatible applications and infrastructures.

IONOS CLOUD Object Storage is included with every contract, with no need for additional registration or activation. Through a user-friendly graphical interface, as well as standard S3-compatible Object Storage clients, customers can efficiently manage their objects and configure access controls using Bucket Policies in accordance with the S3 Object Storage standard.

3.1.4 NAT Gateway (IONOS CLOUD Managed NAT Gateway)

In all locations, IONOS CLOUD provides a Managed Network Address Translation (NAT) Gateway. This service is exposing a Source NAT gateway which means it allows access from the virtual instance to the internet but blocks requests from the internet to the virtual infrastructure. This enables internet access to virtual machines without exposing them to the internet by a public interface. While being "hidden" from the internet and not being exposed to threats, the virtual machine still can initiate a connection to the customizable targets on the internet, e.g., to download new software updates or patches.

3.1.5 DNS (IONOS Cloud DNS)

IONOS Cloud DNS allows customers to publish Domain Name System (DNS) zones for their domains and subdomains on public Name Servers.

Customers can manage their DNS zones and records via the Cloud DNS API and also create and manage Reverse DNS records for IPv4 and IPv6 addresses.

The IONOS CLOUD Name Server infrastructure is distributed across 14 points of presence (POPs) in Europe and the USA to ensure fast and reliable DNS resolution for customers in these locations.

3.1.6 LoadBalancer (IONOS CLOUD Managed Network Load Balancer)

IONOS CLOUD offers a Managed Network Load Balancer (NLB) that is balancing layer 4/ TCP-based network traffic. This service is available in all locations.

Network Load Balancers can be provisioned as a private as well as a public load balancer. A public load balancer requires the configuration of a reserved public IP address for the target configuration. The network load balancer allows the configuration of multiple, individual load balancer rules which can be applied to virtual machines being members of the listener LAN.

3.2 OpenShift Container Platform resource requirements on IONOS CLOUD

3.2.1 Compute

IONOS CLOUD offers different **Compute Engine** services. For OpenShift Container Platform, it is required to choose the **Dedicated Core Server** offerings, as this provides the best performance for the OpenShift Container Platform Nodes.

It is generally recommended to provide sufficient compute resources, especially for the Control Plane Nodes of OpenShift Container Platform.
As we do not have a deeper integration of OpenShift Container Platform to the IONOS CLOUD at this time, we can't use i.e. ControlPlaneMachineSets to easily scale the Control-Plane Nodes as our Cluster grows over time.
For this reason, it is recommended to start with **8 vCPUs and 16 GB RAM** for **Control-Plane Nodes**.

Worker-Nodes should be sized according to the expected application workload but should respect the required minimum for vCPU and Memory. Within this deployment guide, we are using the recommended **4 vCPUs and 8GB RAM for Worker Nodes**.

3.2.2 Storage

OpenShift Container Platform is sensitive to disk performance, and faster storage is recommended, particularly for `etcd` on the Control-Plane Nodes which require a 10 ms p99 fsync duration.
Note that on many cloud platforms, storage size and IOPS scale together, so you might need to over-allocate storage volume to obtain sufficient performance.
Tests have shown that on IONOS CLOUD, **600GB SSD Premium** volumes do have the needed IOPS and latency for running `etcd` and other **Control-Plane components**.
You can learn more about the `etcd` disk backend performance requirements in the following Red Hat Solution: https://access.redhat.com/solutions/4770281

As the disk performance needs for **Worker Nodes** are lower, we can use the recommended **120GB SSD Premium** volume size here.

3.2.3 NTP

OpenShift Container Platform clusters are configured to use a public Network Time Protocol (NTP) server by default. If you want to use a local enterprise NTP server, or if your cluster is being deployed in a disconnected network, you can configure the cluster to use a specific time server. For more information, see the documentation for Configuring chrony time service.

3.2.4 DNS

In OpenShift Container Platform deployments, DNS name resolution is required for the following components:

- The Kubernetes API
- The OpenShift Container Platform application wildcard
- The control plane and compute machines

Reverse DNS resolution is also required for the Kubernetes API, the control plane machines, and the compute machines. It is recommended to use a DHCP server to provide the hostnames to each cluster node.

Please review the official OpenShift documentation to prepare all the needed DNS configurations.

3.2.5 Load Balancing

Before you install OpenShift Container Platform, you must provision the API and application Ingress load balancing infrastructure. In production scenarios, you can deploy the API and application Ingress load balancers separately so that you can scale the load balancer infrastructure for each in isolation.

The load-balancing infrastructure must meet the following requirements:

API load balancer: Provides a common endpoint for users, both human and machine, to interact with and configure the platform. Configure the following conditions:

- Layer 4 load balancing only. This can be referred to as Raw TCP, SSL Passthrough, or SSL Bridge mode. If you use SSL Bridge mode, you must enable Server Name Indication (SNI) for the API routes.
- A stateless load balancing algorithm. The options vary based on the load balancer implementation.

Configure the following ports on both the front and back of the load balancers:

Port	**Back-end machines (pool members)**	**Internal**	**External**	**Description**
6443	Control plane. You must configure the /readyz endpoint for the API server health check probe.	✅	✅	Kubernetes API Server

Port	Back-end machines (pool members)	Internal	External	Description
22623	Control Plane	✅		Machine config server

Application Ingress load balancer: Provides an ingress point for application traffic flowing in from outside the cluster. A working configuration for the Ingress router is required for an OpenShift Container Platform cluster.
Configure the following conditions:

- Layer 4 load balancing only. This can be referred to as Raw TCP, SSL Passthrough, or SSL Bridge mode. If you use SSL Bridge mode, you must enable Server Name Indication (SNI) for the ingress routes.
- A connection-based or session-based persistence is recommended, based on the options available and types of applications that will be hosted on the platform.

Configure the following ports on both the front and back of the load balancers:

Port	Back-end machines (pool members)	Internal	External	Description
443	The machines that run the Ingress Controller pods, compute, or worker, by default.	✅	✅	HTTPS traffic
80	The machines that run the Ingress Controller pods, compute, or worker, by default.	✅	✅	HTTP traffic

Please refer to chapter 3.2.7 to learn more about the IONOS CLOUD specific LoadBalancer implementation and configuration.

Chapter 6 will also provide more details and examples about configuring LoadBalancing for OpenShift on IONOS CLOUD.

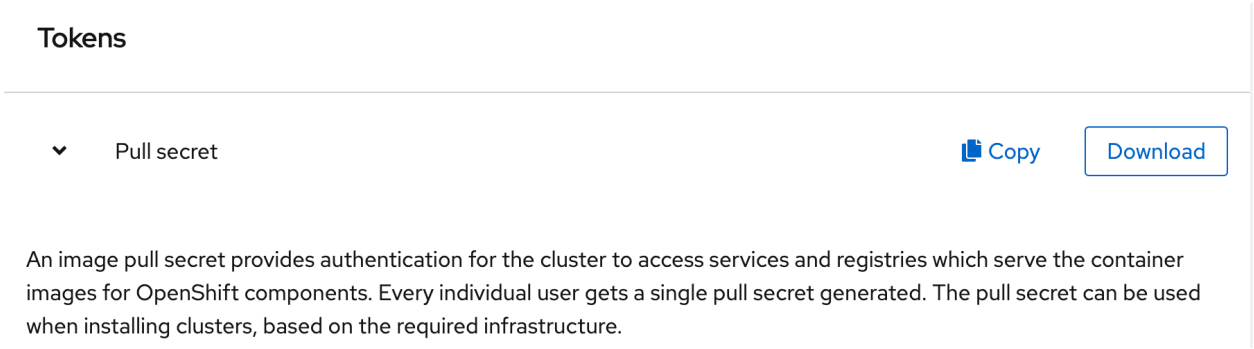
3.3 General prerequisites needed

3.3.1 Pull Secret from console.redhat.com

An image pull secret provides authentication for the cluster to access services and registries that serve the container images for OpenShift components. Every individual user gets a single pull secret generated.

The pull secret is used when installing an OpenShift Container Platform cluster.

It can be retrieved from the OpenShift Cluster Manager within the Red Hat Hybrid Cloud Console. Go to Downloads in OpenShift Cluster Manager and find your pull secret in the **Tokens** category.



Important: Do not share your pull secret. The pull secret should be treated like a password.

3.3.2 SSH-Key

During an OpenShift Container Platform installation, you can provide an SSH public key. This key is added to the `~/.ssh/authorized_keys` list for the “core” user on each Red Hat Enterprise Linux CoreOS (RHCOS) node, enabling password-less SSH access. You can then use the corresponding private key to SSH into the nodes as the user "core."

If you do not already have an existing SSH key pair, please create one according to the following example:

```
ssh-keygen -t ed25519 -C 'ocp-admin@ionos' -f <path>/<file_name>
```

For seamless SSH access to the Red Hat Enterprise Linux CoreOS (RHCOS) nodes, it is recommended to add the private SSH key to your local user's SSH agent.

More detailed information can be found in the OpenShift documentation.

3.3.3 Platform selection

OpenShift Container Platform includes the Cluster Cloud Controller Manager Operator (CCCMO). CCCMO is based on Kubernetes and the overall idea is to move the cloud controller functionality out of the Kubernetes core and into separate Cloud Controller Managers.

A Cloud Controller Manager is a Kubernetes component that allows cloud providers to integrate their services with Kubernetes clusters. It manages node lifecycle, load balancers, persistent volumes, and other cloud-specific resources for some of the most popular hyperscaler Cloud Providers.

However, your preferred sovereign cloud provider IONOS CLOUD currently has no Cloud Controller Manager integration for OpenShift Container Platform.For this reason, we are going to deploy and configure some of the infrastructure requirements (i.e. LoadBalancer, Networking, Storage) manually throughout this documentation.

For this reason, it is important to select the appropriate `platform: external` configuration directive (in the `install-config.yaml` file) during the OpenShift Container Platform installation on IONOS CLOUD.

The deployment chapters within this document will honor the `platform: external` requirement.

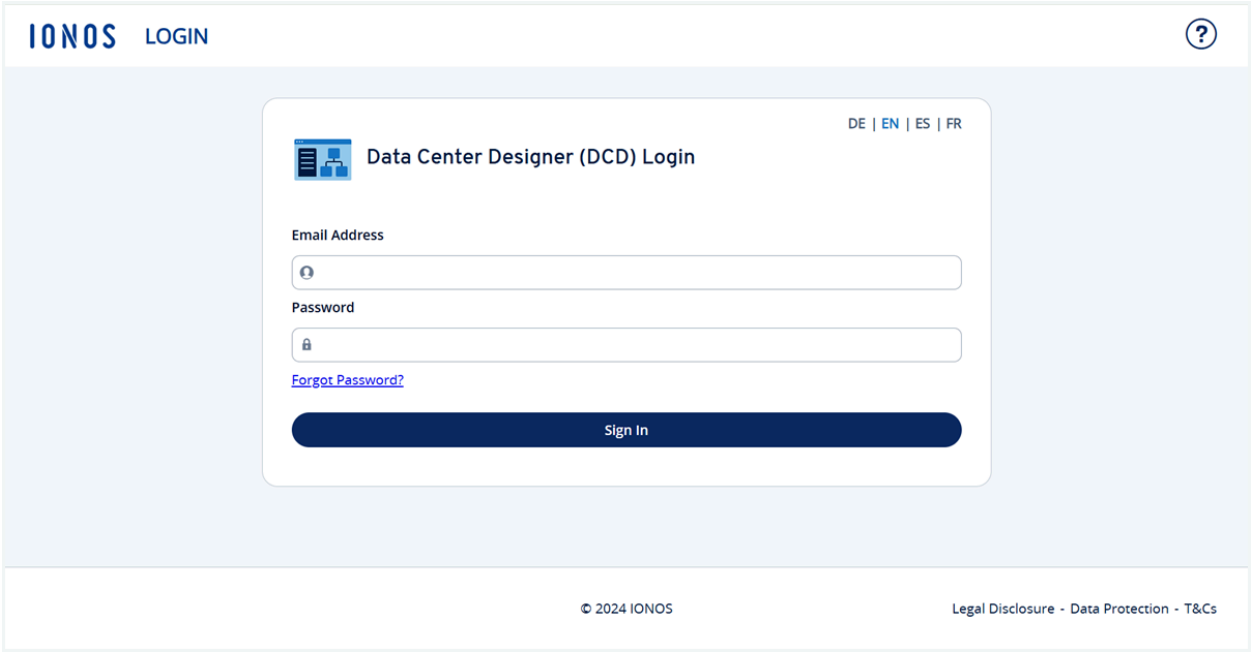
Please carefully inspect the `install-config.yaml` template in Chapter 6.4.1, which should be used for the installation.

You can find more details about the OpenShift Container Platform type “External” in the following blog post.

4 Prepare general resources for the IONOS CLOUD

4.1 Log in to the IONOS CLOUD Data Center Designer (DCD)

- Open a web browser and navigate to https://dcd.ionos.com.
- On the top right corner of the login screen, select your preferred language. The DCD supports German (DE), English (EN), French (FR), and Spanish (ES) languages.
- Enter the **Email Address** and **Password** details that were obtained during the sign-up process for an IONOS CLOUD account.

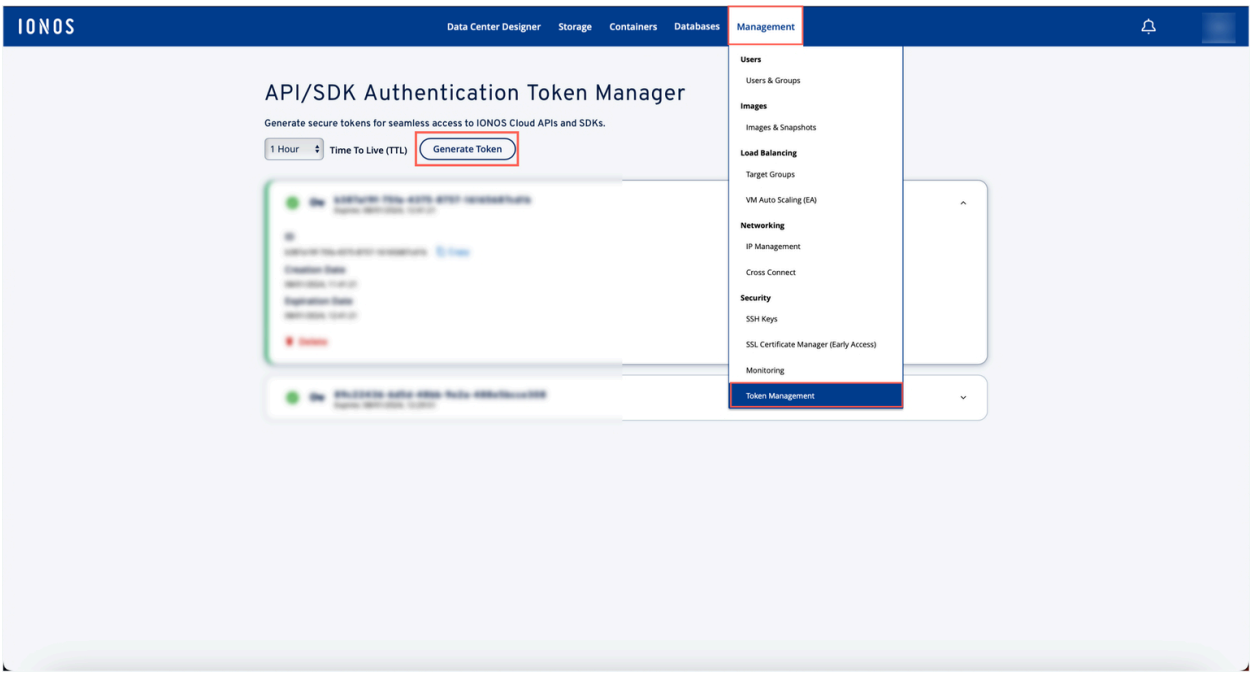


4.2 Generate authentication token

As we are going to create the resources on IONOS CLOUD with “Terraform”, it is recommended to generate and use an authentication token when interacting with the IONOS CLOUD API. For IONOS CLOUD accounts with two-factor authentication enabled (which is always recommended!), it’s mandatory to use authentication tokens.

To create a secure authentication token for authorizing to use APIs and SDKs, follow these steps:

- In the **DCD**, go to **Menu > Management > Token Management**



- In the **API/SDK Authentication Token Manager**, select **Generate Token**
- Press **Download** to save the **Token** to a file and **Close** the **token-generated** window.

4.3 Adding SSH Key

The **DCD's SSH Keys view** allows you to save and manage up to 100 public SSH keys for SSH access setup. SSH Keys are bound to your IONOS CLOUD account and not to a specific VDC. Configuring your public SSH key as a **Default key**, it will be automatically pre-selected and used for IONOS CLOUD VMs deployed.

You may refer to Chapter 3.4.2 to learn more about generating SSH Key Pairs.

However, keep in mind that the SSH Key added to the IONOS CLOUD DCD will not be automatically injected into the Red Hat CoreOS Nodes deployed by the OpenShift Container Platform installation described in this deployment guide.

For this reason, it is recommended to use one SSH Key Pair (i.e. named **vdc-admin**) for getting access to the “helper” VM “Management” and a second one (i.e. named **ocp-admin**) provided to the OpenShift Container Platform installation process for accessing the Red Hat CoreOS Nodes during i.e. troubleshooting.

Please refer to the IONOS CLOUD documentation about how to add the “**vdc-admin**” SSH private key as a default key to the DCD.

4.4 Reserve public IPv4 addresses

For the architecture described within this deployment guide, we will need three public IP addresses on the IONOS CLOUD:

- Public IP address for the “Management” Red Hat Enterprise Linux host
- Public IP address for the IONOS CLOUD Network-Load-Balancer (API and Ingress traffic for OpenShift Container Platform)
- Public IP address for the IONOS CLOUD NAT Gateway Egress

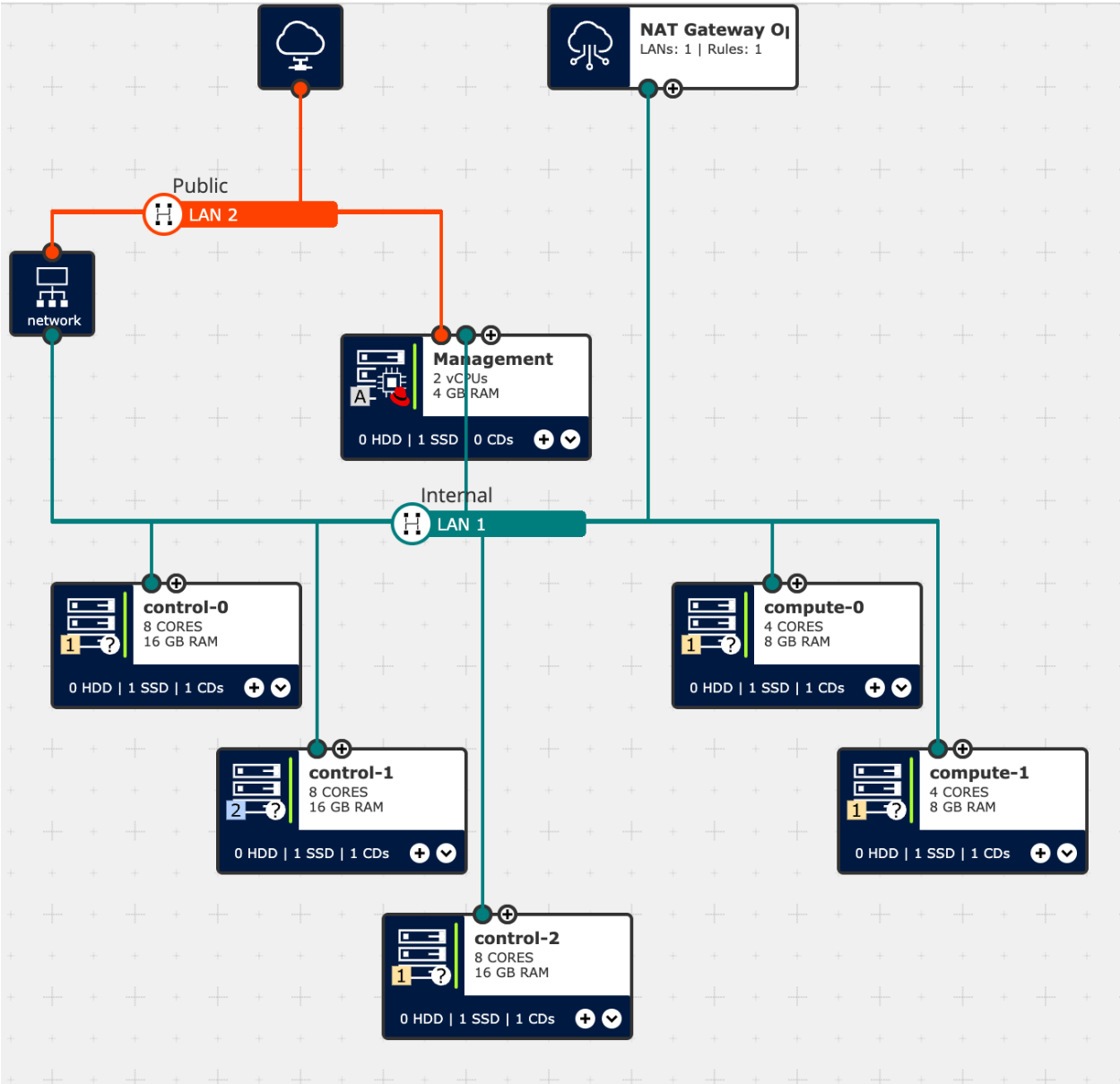
To reserve an IP address for the “**Management**” host, follow these steps:

- In the **DCD**, go to the **Menu** > **Network Services** > **IP Management.**
- In the **IP Manager**, select **+ Reserve IPs**.
- Enter the following IP block information:
    - Name: Enter a name for the IP block, **here “Management”**
    - Number of IPs: Enter the number of IPv4 addresses you want to reserve, **here: 1**
    - Region: Enter the location of the IONOS CLOUD data center where you want your IPs to be available, here: **Germany/Frankfurt am Main**
- Confirm your entries by selecting **Reserve IPs**.

Repeat this process for the “**Load Balancer**” and **“NAT Gateway”** public IP address.

5. Architectural overview

Before we are finally preparing the IONOS CLOUD resources and start the OpenShift Container Platform deployment in Chapter 6, let's have a look at the desired architectural overview:



This screenshot from the IONOS CLOUD Data Center Designer (DCD) shows an overview of the Cloud resources to be deployed:

- Internal and Public Network (LAN 1 and LAN 2)
- Management Server connected to both networks
- Network Load Balancer and Management Server connected to LAN 2 with Internet Access
- NAT Gateway with SNAT Rule, providing outgoing internet access for all Dedicated Core

Server VMs in LAN 1

- Three OpenShift Container Platform Control-Plane Nodes (named control-0 to -2) with 8 vCPUs and 16GB RAM
- Two OpenShift Container Platform Worker Nodes (named compute-0 to -1) with 4 vCPUs and 8GB RAM

6 Deploy Openshift Container Platform with the Agent-based Installer on IONOS CLOUD

6.1 General description

6.1.1 About the Agent-based Installer

OpenShift Container Platform's agent-based installation method offers significant benefits for deploying clusters, particularly in on-premises and disconnected environments. This modern approach combines flexibility with simplicity while addressing key operational challenges.

6.1.2 Understanding the Agent-based Installer

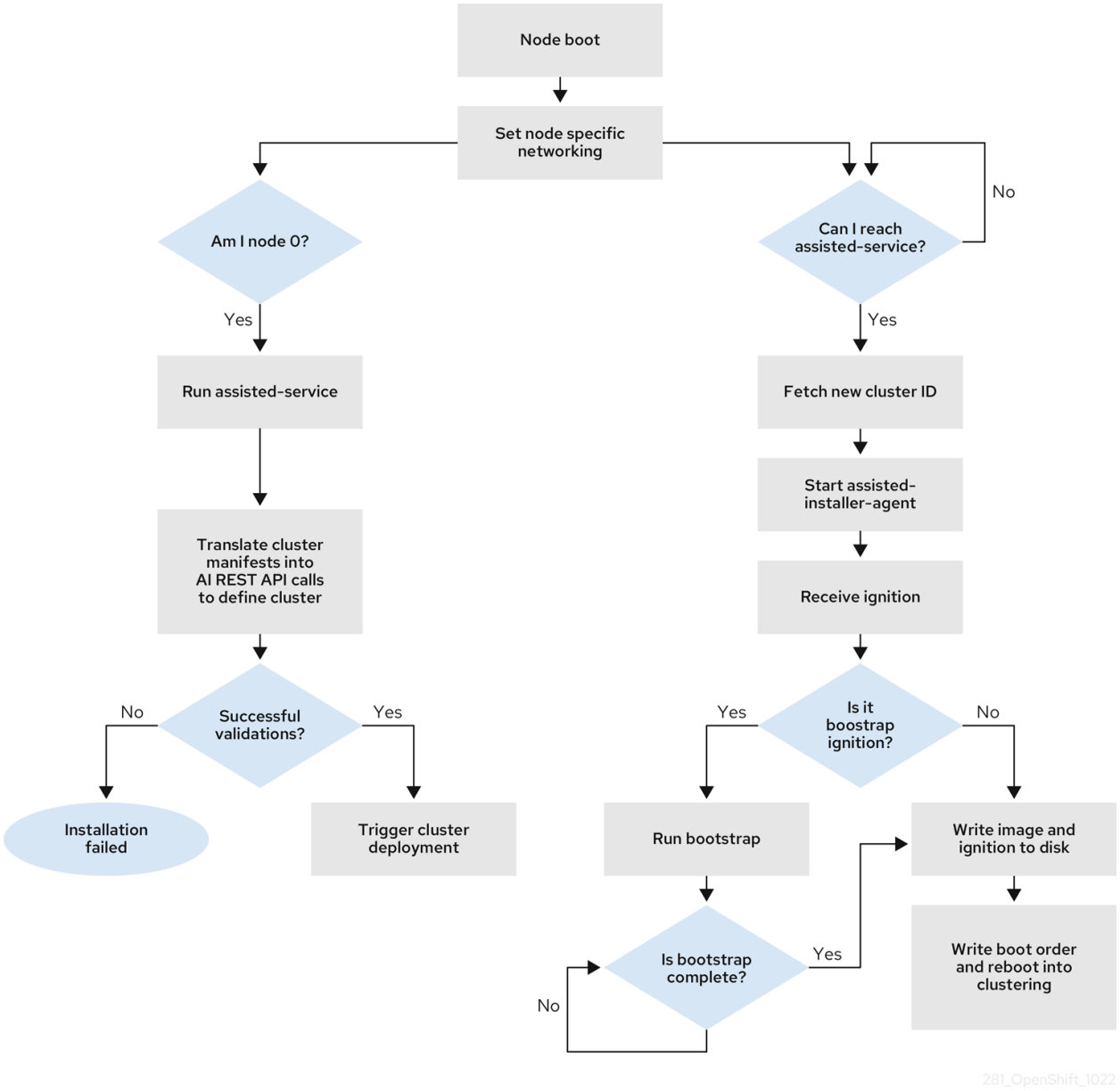
Agent-based installation is a subcommand of the OpenShift Container Platform installer. The `openshift-install agent create image` subcommand generates a bootable ISO image containing all of the information required to deploy an OpenShift Container Platform cluster, with an available release image.

The configuration is in the same format as for the installer-provisioned infrastructure and user-provisioned infrastructure installation methods.

6.1.3 Agent-based Installer Workflow

One of the control plane hosts runs the Assisted Service at the start of the boot process and eventually becomes the bootstrap host. This node is called the **rendezvous host** (node 0).
The Assisted Service ensures that all the hosts meet the requirements and triggers an OpenShift Container Platform cluster deployment.
All the nodes have the Red Hat Enterprise Linux CoreOS (RHCOS) image written to the disk.
The non-bootstrap nodes reboot and initiate a cluster deployment. Once the nodes are rebooted, the rendezvous host reboots and joins the cluster.
The bootstrapping is complete and the cluster is deployed.

The following diagram shows the installation workflow in detail:



You can find more details about the Agent-based Installer in the OpenShift documentation.

Please remember: While the Agent-based Installer generally supports different OpenShift Cluster topologies, only the default OpenShift High Availability Cluster (consisting of 3 Control-Plane Nodes and 2 Worker Nodes) topology is relevant, as this is the bare minimum recommended for production environments **and** required for the Red Hat Ecosystem certification.

6.2 Check Prerequisites

Before actually starting the deployment of OpenShift Container Platform, make sure that the following prerequisites are fulfilled:

- Pull Secret - see Chapter 3.5.1
- SSH-Key “ocp-admin” - see Chapter 3.5.2
- SSH-Key “vdc-admin” added to the IONOS CLOUD DCD account - see Chapter 4.3
- IONOS CLOUD API Token - see Chapter 4.2
- Terraform and terraform-provider-ionoscloud >= 6.4.10

6.3 Create the IONOS CLOUD VDC and the Management Server

To install, configure, and manage OpenShift Container Platform from a secured virtual machine within the VDC, we are going to prepare a small vCPU Server (2 vCPU, 4GB RAM, 30GB SSD) using a Red Hat Enterprise Linux (RHEL) 9 image from the IONOS CLOUD catalog.

This vCPU Server will be created along with the IONOS CLOUD VDC and the needed networks using “Terraform”.

6.3.1 Running Terraform locally to create initial IONOS CLOUD resources

Assuming you have “Terraform” installed on your local (Linux) machine, please follow the following steps:

- Create IONOS CLOUD Token environment variable for “Terraform”
  In Chapter 4.2, we generated the IONOS CLOUD API token that we are now going to use with “Terraform”.

```
None
# export IONOS_TOKEN="your-IONOS-API-TOKEN"
```

- Create infrastructure directory

```
None
# mkdir $HOME/infrastructure
```

- Create “Terraform” `main.tf`

In “Terraform”, `main.tf` is a commonly used file that serves as the primary configuration file for defining infrastructure resources.
Create the `main.tf` file with the following content:

```
None
# Configuring the provider for IONOS CLOUD
terraform {
  required_providers {
    ionoscloud = {
      source  = "ionos-cloud/ionoscloud"
      version = ">= 6.4.10"
    }
  }
}

# Creating the IONOS CLOUD VDC in Germany/Frankfurt am Main location
resource "ionoscloud_datacenter" "openshift" {
  name        = "OpenShift Reference Cluster"
  location    = "de/fra/2"
  description = "OpenShift Container Platform on IONOS CLOUD"
}

# Configuring public LAN
resource "ionoscloud_lan" "public" {
  datacenter_id = ionoscloud_datacenter.openshift.id
  public        = true
  name          = "Public"
}

# Configuring private LAN
resource "ionoscloud_lan" "internal" {
  datacenter_id = ionoscloud_datacenter.openshift.id
  public        = false
  name          = "Internal"
  depends_on = [
    ionoscloud_lan.public,
  ]
}

# Creating the Management Server in the VDC
resource "ionoscloud_vcpu_server" "management" {
    name                  = "Management"
    datacenter_id         = ionoscloud_datacenter.openshift.id
    cores                 = 2
    ram                   = 4096
```

```
    image_name            = "rhel:9"
    ssh_keys              = ["paste-your-public-ssh-key-called-vdc-admin"]
    volume {
        name              = "system"
        size              = 30
        disk_type         = "SSD Standard"
        bus               = "VIRTIO"
    }
    nic {
        lan               = ionoscloud_lan.public.id
        name              = "public-nic"
        dhcp              = true
        ips               = [ var.management_server_ip[0] ]
    }
}

# Adding a second NIC to the Management Server for the internal LAN
resource "ionoscloud_nic" "management" {
  datacenter_id         = ionoscloud_datacenter.openshift.id
  server_id             = ionoscloud_vcpu_server.management.id
  lan                   = ionoscloud_lan.internal.id
  name                  = "internal-nic"
  dhcp                  = true
}
```

- Create “Terraform” variables.tf
  In “Terraform”, variables are defined in a `variables.tf` file using the `variable` block.
  Please change the “management_server_ip” to the Public IP reserved in Chapter 4.4
  Create the `variables.tf` with the following content:

```
None
# As we need to call Terraform multiple times due to dependencies between the
different IONOS CLOUD resources,
# it makes sense to have on central variables.tf file.
# There are new variables which need to be added in every stage, so please
carefully check the comments.

## STAGE 1: Create datacenter, LANs and Management host

# External IP for the management host, as reserved in the IONOS CLOUD DCD
IP-Management tool
```

```
variable "management_server_ip" {
  description = "External IP Management Server"
  type        = list(any)
  default     = ["xxx.xxx.7.70"]
}
```

- “Terraform” stages to be executed:
  - Change into the `$HOME/infrastructure` directory
  - **Initialize** (`terraform init`)
    - Prepares the working directory by downloading provider plugins and configuring backends.
  - **Plan** (`terraform plan`)
    - Creates an execution plan, showing what changes will be made to match the desired state.
  - **Apply** (Deploy) (`terraform apply`)
    - Executes the plan to create, update, or destroy resources as needed.

- The “Terraform” Apply step should print out the following output:

```
None
datacenter-id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
internal-lan-cidr = "10.7.200.0/23"
internal-lan-id = "1"
public-lan-id = "2"
management-internal-ip = tolist(["10.7.200.11",])
management-public-ip = "xxx.xxx.7.70"
```

Those IONOS CLOUD resource IDs and IP addresses will be needed in the next steps as input variables, so please make sure to note them down.

6.3.2 Preparing the “Management” virtual machine

While working through Chapter 6.3.1 earlier, we created a small Red Hat Enterprise Linux virtual machine on IONOS CLOUD.
The purpose of this VM is to act as the management environment for the OpenShift Container Platform Cluster to be installed.
You are going to connect to this machine externally via SSH key. The “Management” VM acts as some kind of “jump-host” to the OpenShift Container Platform Nodes in case of emergency access needed.

We will also utilize the “Management” VM to run the OpenShift Installer, prepare the ISO images, host additional boot artifacts and eventually connect to additional systems in the specific VDC.

6.3.3 Connecting to the “Management” virtual machine

When preparing the general IONOS CLOUD resources in Chapter 4, you also added an SSH Public Key for the Red Hat Enterprise Linux instance created in Chapter 6.3.1.

On your local machine, please add the corresponding private SSH Key to your SSH Agent for passwordless access:

```
None
$ eval "$(ssh-agent -s)"
Agent pid 31874


$ ssh-add <path>/vdc-admin_ssh_priv_key>

Identity added: /home/<you>/<path>/<file_name> (<computer_name>)
```

After that, you should be able to connect to the “Management” VM with the public IP address assigned earlier:

```
None
$ ssh root@management-vm-public-ip
```

Update the VM packages and reboot the VM afterwards:

```
None
$ dnf update -y
...
python3-resolvelib-0.5.4-5.el9.noarch

Complete!
$ reboot
$ Connection to xxx.xxx.xxx.xxx closed by remote host.
```

Connect again:

```
None
$ ssh root@management-vm-public-ip
```

6.3.4 Installing required software

Note: Within this deployment guide and to keep the guide shorter, we are working as the “root” user on the “Management” VM. This (of course) isn’t the best practice, and you should consider creating a dedicated unprivileged user for the following activities. It’s also important to harden and secure the “Management” VM according to your organization's individual security requirements and standards!

In the first step we installing some needed and helpful additional tools via the RPM packages available for the RHEL server:

```
None
# dnf -y install nmstate jq bind-utils tmux wget bash-completion httpd git
```

Enable the Apache Http-Server as we will need it to serve boot artifacts later on:

```
None
# systemctl enable httpd.service
# systemctl start httpd
```

As we will later need the Apache Http-Server to host the root-fs image for Red Hat CoreOS, please add http as allowed service to the default IONOS CLOUD RHEL image configuration:

```
None
# firewall-cmd --permanent --zone=public --add-service=http && firewall-cmd
--reload
```

Note:
It is recommended to create a more sophisticated RHEL firewall configuration for the “Management” VM (i.e. allowing http and other services only on the internal interface), but this is out of scope within this OpenShift Deployment Guide.

Red Hat Enterprise Linux (RHEL) does not ship with HashiCorp Terraform pre-installed. However, you can easily install Terraform on RHEL using the yum or dnf package manager by adding the HashiCorp repository to your system:

```
None
# dnf config-manager addrepo --add-repo
https://rpm.releases.hashicorp.com/RHEL/hashicorp.repo
```

```
# dnf -y install terraform
```

As a last step, we need to download and install the OpenShift Container Platform command line client (oc) and the Installer.

NOTE: Carefully inspect the `openshift-install-linux` URL, as this specifies the actual Red Hat OpenShift Container Platform release you are going to install.

```
None
# mkdir /root/bin && wget -qO-
https://mirror.openshift.com/pub/openshift-v4/x86_64/clients/ocp/4.17.27/openshift
-install-linux-4.17.27.tar.gz | tar -xz -C /root/bin

# wget -qO-
https://mirror.openshift.com/pub/openshift-v4/x86_64/clients/ocp/stable/openshift-
client-linux-amd64-rhel9.tar.gz | tar -xz -C /root/bin
```

This will download and extract the needed binary files in the /root/bin directory, which should be in your PATH already:

```
None
# ll /root/bin
total 1006020
-rw-r--r--. 1 root root       950 Feb 26 06:38 README.md
-rwxr-xr-x. 2 root root 185054328 Feb 26 06:38 kubectl
-rwxr-xr-x. 2 root root 185054328 Feb 26 06:38 oc
-rwxr-xr-x. 1 root root 660041880 Mar 11 20:15 openshift-install
```

6.4 Install a Highly Available OpenShift Container Platform Cluster

Within this chapter we are going to the detailed steps to configure the Cluster specific resources on IONOS CLOUD and finally install the OpenShift Container Platform Cluster using the Agent-based Installer.

We are assuming the following names:

- Cluster Name: ocp
- Base Domain: ionos.yourdomain.de

All the following steps will be executed from the "Management" server we installed and configured in Chapter 6.3.

6.4.1 Prepare the “Terraform” working directories on the “Management” server

- Create API Token environment variable for “Terraform”
  In Chapter 4.2, we generated the IONOS CLOUD API token that we are now going to use with “Terraform”.

```
None
# export IONOS_TOKEN="your-IONOS-API-TOKEN"
```

- Create directory structure

```
None
# mkdir $HOME/02-provision-infrastructure $HOME/03-configure-lb
```

- Create the `$HOME/variables.tf` file with the following content:

```
None
## STAGE 2: Create VMs and NAT Gateway

# IONOS CLOUD Datacenter ID (see Terraform output from STAGE 1 and set
accordingly)
variable "datacenter_id" {
  default   = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
  description = "imported IONOS CLOUD Datacenter ID"
}

# IONOS CLOUD generated CIDR for the internal LAN (see Terraform output from STAGE
1 and set accordingly)
variable "internal_lan_cidr" {
  default     = "10.7.200.0/23"
  description = "CIDR for the internal/private network."
```

```
}

# IONOS CLOUD generated ID number of the internal LAN (see Terraform output from
STAGE 1 and set accordingly)
variable "internal_lan_id" {
  default   = "1"
  description = "ID of the internal LAN"
}

# IONOS CLOUD generated ID number of the public LAN (see Terraform output from
STAGE 1 and set accordingly)
variable "public_lan_id" {
  default   = "2"
  description = "ID of the public LAN"
}

# External IP for the NAT Gateway reserved earlier as described in the Deployment
Guide
variable "nat_egress_ip" {
  description = "External IP for NAT Gateway, see IONOS CLOUD DCD IP-Management
Tool"
  default     = "xx.xxx.132.76"
}

## STAGE 3: Create the IONOS CLOUD Network Load Balancer (NLB) for OpenShift API
and Ingress

# External IP for the NLB reserved earlier as described in the Deployment Guide
variable "external_lb_ip" {
  description = "External LoadBalancer IP API/Ingress"
  type        = string
  default     = "xxx.xxx.96.90"
}

# IP addresses for the Control Plane Nodes (see Terraform output from STAGE 2 and
set accordingly)
variable "control-ips" {
  type    = list(any)
  default = ["10.7.200.xx", "10.7.200.xx", "10.7.200.xx"]
}

# IP addresses for the Compute Nodes (see Terraform output from STAGE 2 and set
accordingly)
variable "compute-ips" {
  type    = list(any)
```

```
  default = ["10.7.200.xx", "10.7.200.xx"]
}
```

6.4.2 Provision IONOS CLOUD Dedicated Core Servers and IONOS CLOUD NAT Gateway

As IONOS CLOUD offers the freedom to individually configure every compute resource to your needs, there are no general instance types available for the Dedicated Core Server product.

Therefore, you need to follow the resource configuration (Compute and SSD storage) outlined within this chapter, as this is exactly the required minimum production configuration that was used for the Red Hat Ecosystem certification.

In this step, we are provisioning the needed VMs for the Control-Plane and Worker OpenShift Nodes. As we need to rely on DHCP within the private network we created in the VDC previously, it's needed to provision the VMs before we actually create the bootable ISO for the Agent-based OpenShift installation.
Once the VM is provisioned, we have the DHCP assigned internal IP address we are going to use for the static IP configuration of the OpenShift RHCOS Nodes.
The static IP configuration is needed due to the limited IONOS CLOUD DHCP functionality for private networks.

Additionally, we are also creating the IONOS CLOUD NAT Gateway and a general SNAT rule to connect our private VMs to public software repositories and NTP (Network Time Protocol) servers. Please note that you might need a more fine-grained SNAT rule, according to your organization's individual security requirements and standards.

- Create “Terraform” `$HOME/02-provision-infrastructure/main.tf` with the following content:

```
None
# Configuring the provider for IONOS CLOUD
terraform {
  required_providers {
    ionoscloud = {
      source  = "ionos-cloud/ionoscloud"
      version = ">= 6.4.10"
    }
  }
}

# Creating the Control Plane Nodes
resource "ionoscloud_server" "control" {
  count         = 3
  name          = "control-${count.index}"
  datacenter_id = var.datacenter_id
  cores         = 8
  ram           = 16384
  cpu_family    = "AMD_TURIN"

  volume {
    name      = "control-${count.index}-storage"
    size      = 600
    disk_type = "SSD Premium"
    bus       = "VIRTIO"
    licence_type  = "OTHER"
  }

  nic {
    lan  = var.internal_lan_id
    name = "internal_nic"
    dhcp = true
    mac  = "02:01:e1:40:fe:5${count.index}" #setting a predictable MAC address, as
we need this in the AgentCongfig.yaml
  }
}

# Creating the Compute Nodes

resource "ionoscloud_server" "compute" {
  count         = 2
  name          = "compute-${count.index}"
```

```
  datacenter_id = var.datacenter_id
  cores         = 4
  ram           = 8192
  cpu_family    = "AMD_TURIN"

  volume {
    name      = "compute-${count.index}-storage"
    size      = 120
    disk_type = "SSD Premium"
    bus       = "VIRTIO"
    licence_type  = "OTHER"
  }

  nic {
    lan  = var.internal_lan_id
    name = "internal_nic"
    dhcp = true
    mac  = "02:01:e1:50:fe:5${count.index}" #setting a predictable MAC address, as
we need this in the AgentCongfig.yaml
  }
}

# Creating the NAT Gateway
resource "ionoscloud_natgateway" "openshift" {
  datacenter_id = var.datacenter_id
  name          = "NAT Gateway OpenShift"
  public_ips    = [var.nat_egress_ip]
  lans {
    id          = var.internal_lan_id
    gateway_ips = [cidrhost(var.internal_lan_cidr,1)]
  }
}

## Adding NAT Gateway SNAT Rule for Internet Access
resource "ionoscloud_natgateway_rule" "internet_access" {
  datacenter_id = var.datacenter_id
  natgateway_id = ionoscloud_natgateway.openshift.id
  name          = "internet access"
  type          = "SNAT"
  source_subnet = var.internal_lan_cidr
  public_ip     = var.nat_egress_ip
}
```

- Create “Terraform” `$HOME/02-provision-infrastructure/outputs.tf` with the following content:

```
None
output "nat-gateway-internal-ip" {
  value = ionoscloud_natgateway.openshift.lans[0].gateway_ips
  description = "Internal IP of the NAT Gateway, needs to be the default GW for
all VMs"
}

output "control-plane-ips" {
  value = [for control in ionoscloud_server.control :
          format(
          "Name: %s | IP: %s",
          control.name,
          control.primary_ip
          )]
  description = "IP address for the VM, needed for static IP config in
agent-config.yaml"
}

output "compute-node-ips" {
  value = [for compute in ionoscloud_server.compute :
          format(
          "Name: %s | IP: %s",
          compute.name,
          compute.primary_ip
          )]
  description = "IP address for the VM, needed for static IP config in
agent-config.yaml"
}
```

- Create symbolic link to the `variables.tf` file created earlier:

```
None
# ln -s $HOME/variables.tf $HOME/02-provision-infrastructure/variables.tf
```

- Edit the `variables.tf` file and set the following details for “Stage 2”:
    - Datacenter ID
    - CIDR for the internal LAN in the
- Apply “Terraform”:

```
None
# cd $HOME/02-provision-infrastructure/
# terraform init
# terraform plan
# terraform apply
```

- The “Terraform” Apply step should print the following output:

```
None
compute-node-ips = [
  "Name: compute-0 | IP: 10.7.200.xx",
  "Name: compute-1 | IP: 10.7.200.xx",
]
control-plane-ips = [
  "Name: control-0 | IP: 10.7.200.xx",
  "Name: control-1 | IP: 10.7.200.xx",
  "Name: control-2 | IP: 10.7.200.xx",
]
nat-gateway-internal-ip = tolist([
  "10.7.200.1",
])
```

6.4.3 Provision IONOS CLOUD Network Load Balancer for OpenShift

The Managed Network Load Balancer (NLB) is a pre-configured IONOS CLOUD VDC element that provides connection-based layer 4 load balancing features and functionality.
To fulfill the OpenShift Load Balancing requirements outlined in Chapter 3.3.5, we are going to deploy the NLB with Terraform in this step.

We can only configure the NLB forwarding rules once we have the “target” IP addresses of the Control-Plane and Worker Nodes from the previous step.

- Create “Terraform” `$HOME/03-configure-lb/main.tf` with the following content:

None

```
# Configuring the provider for IONOS CLOUD
terraform {
  required_providers {
    ionoscloud = {
      source  = "ionos-cloud/ionoscloud"
      version = ">= 6.4.10"
    }
  }
}

# Setting LB IP locally
locals {
  internal_lb_ip = cidrhost(var.internal_lan_cidr, 5)
}

# LoadBalancer for API and Ingress starts here
resource "ionoscloud_networkloadbalancer" "openshift" {
  datacenter_id  = var.datacenter_id
  name           = "Load Balancer for OpenShift API and Ingress"
  listener_lan   = var.public_lan_id
  target_lan     = var.internal_lan_id
  ips            = [var.external_lb_ip]
  lb_private_ips = ["${local.internal_lb_ip}/23"]
}

# Configuration of IONOS CLOUD Network Loadbalancer Forwarding rules
resource "ionoscloud_networkloadbalancer_forwardingrule" "api-ext" {
  datacenter_id          = var.datacenter_id
  networkloadbalancer_id = ionoscloud_networkloadbalancer.openshift.id
  name                   = "api-ext"
  algorithm              = "ROUND_ROBIN"
  protocol               = "TCP"
  listener_ip            = var.external_lb_ip
  listener_port          = "6443"
  health_check {
    target_timeout = 30000
  }

  dynamic "targets" {
    for_each = var.control-ips
    content {
      ip     = targets.value
      port   = "6443"
      weight = "1"
      health_check {
```

```
        check          = true
        check_interval = 10000
        maintenance    = false
      }
    }
  }
}

resource "ionoscloud_networkloadbalancer_forwardingrule" "api-int" {
  datacenter_id          = var.datacenter_id
  networkloadbalancer_id = ionoscloud_networkloadbalancer.openshift.id
  name                   = "api-int"
  algorithm              = "ROUND_ROBIN"
  protocol               = "TCP"
  listener_ip            = local.internal_lb_ip
  listener_port          = "6443"
  health_check {
    target_timeout = 30000
  }

  dynamic "targets" {
    for_each = var.control-ips
    content {
      ip     = targets.value
      port   = "6443"
      weight = "1"
      health_check {
        check          = true
        check_interval = 10000
        maintenance    = false
      }
    }
  }
}

resource "ionoscloud_networkloadbalancer_forwardingrule" "ignition-int" {
  datacenter_id          = var.datacenter_id
  networkloadbalancer_id = ionoscloud_networkloadbalancer.openshift.id
  name                   = "ignition-int"
  algorithm              = "ROUND_ROBIN"
  protocol               = "TCP"
  listener_ip            = local.internal_lb_ip
  listener_port          = "22623"
  health_check {
    target_timeout = 30000
```

```
  }

  dynamic "targets" {
    for_each = var.control-ips
    content {
      ip     = targets.value
      port   = "22623"
      weight = "1"
      health_check {
        check          = true
        check_interval = 10000
        maintenance    = false
      }
    }
  }
}

resource "ionoscloud_networkloadbalancer_forwardingrule" "ingress80" {
  datacenter_id          = var.datacenter_id
  networkloadbalancer_id = ionoscloud_networkloadbalancer.openshift.id
  name                   = "api"
  algorithm              = "SOURCE_IP"
  protocol               = "TCP"
  listener_ip            = var.external_lb_ip
  listener_port          = "80"
  health_check {
    target_timeout = 30000
  }

  dynamic "targets" {
    for_each = var.compute-ips
    content {
      ip     = targets.value
      port   = "80"
      weight = "1"
      health_check {
        check          = true
        check_interval = 10000
        maintenance    = false
      }
    }
  }
}

resource "ionoscloud_networkloadbalancer_forwardingrule" "ingress443" {
```

```
  datacenter_id          = var.datacenter_id
  networkloadbalancer_id = ionoscloud_networkloadbalancer.openshift.id
  name                   = "api"
  algorithm              = "SOURCE_IP"
  protocol               = "TCP"
  listener_ip            = var.external_lb_ip
  listener_port          = "443"
  health_check {
    target_timeout = 30000
  }

  dynamic "targets" {
    for_each = var.compute-ips
    content {
      ip     = targets.value
      port   = "443"
      weight = "1"
      health_check {
        check          = true
        check_interval = 10000
        maintenance    = false
      }
    }
  }
}
```

- Create “Terraform” `$HOME/03-configure-lb/outputs.tf` with the following content:

```
None
output "nlb-internal-ip" {
  value = ionoscloud_networkloadbalancer.openshift.lb_private_ips
  description = "IP address of the NLB in private/internal network, used for
api-int DNS entry"
}
```

- Create symbolic link to the `variables.tf` file created earlier:

```
None
# ln -s $HOME/variables.tf $HOME/03-configure-lb/variables.tf
```

- Edit the `variables.tf` file and set the following details for “Stage 3”:
  - External Loadbalancer IP as reserved in IONOS CLOUD IP Management
  - Control-Plane Node IPs
  - Computer Node IPs

- Apply “Terraform”:

```
None
# cd $HOME/03-configure-lb/
# terraform init
# terraform plan
# terraform apply
```

- The “Terraform” Apply step should print the following output:

```
None
nlb-internal-ip = tolist([
  "10.7.200.5",
])
```

6.4.4 Create DNS Domain or Subdomain

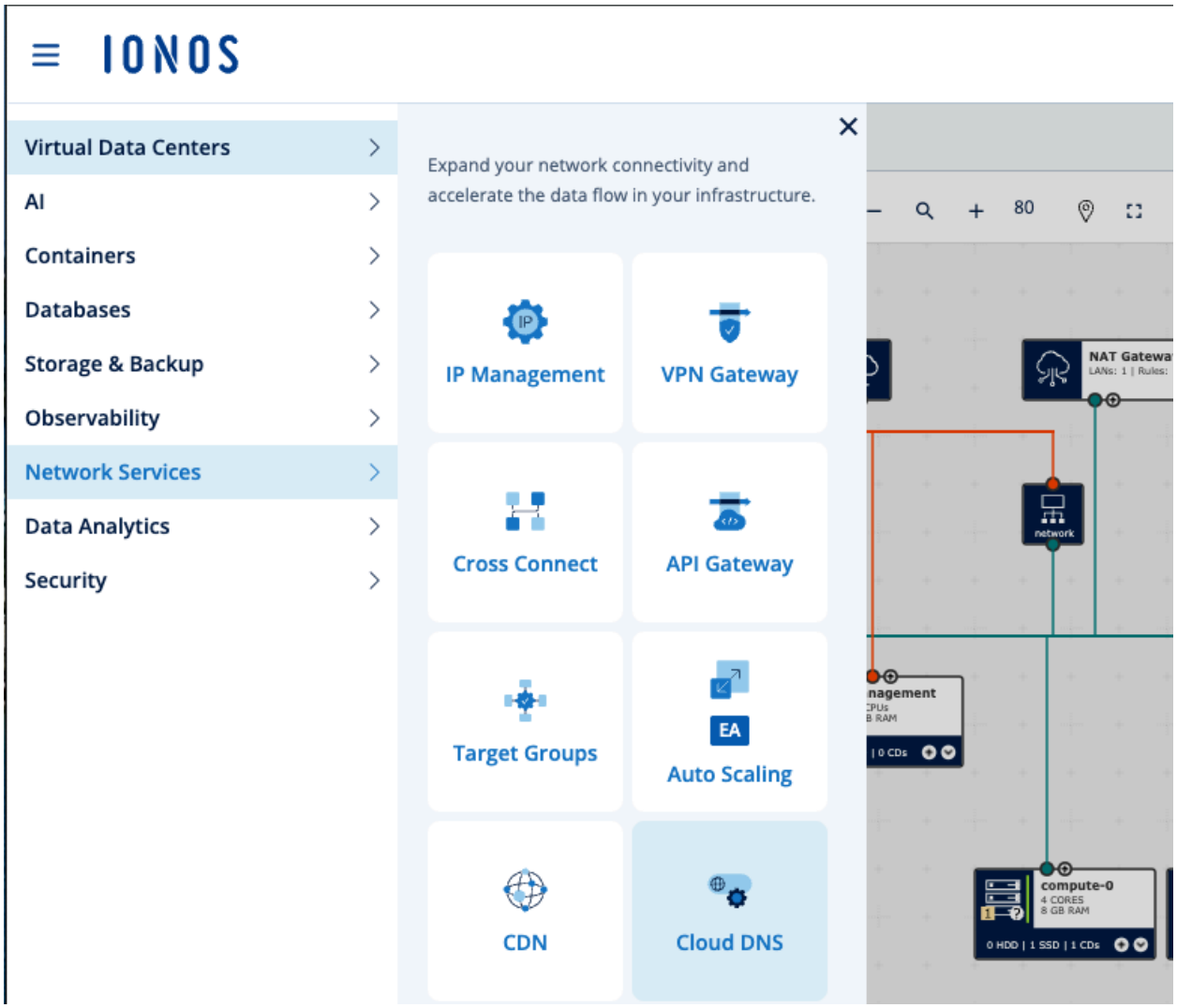
In OpenShift Container Platform, DNS configuration for the API Server, Application Wildcard Ingress and the Nodes in the Cluster is required.

This deployment guide assumes that you have a DNS domain registered with your Registrar. Within the DNS configuration at your Registrar, configure a subdomain which is using the “NS” record type to point to the IONOS Cloud DNS Nameservers.

With IONOS CLOUD Domain Name System (DNS), you can publish your domain names to the global DNS. The feature is built around the concept of DNS zones and records that can be managed through both the Cloud DNS API and the Data Center Designer (DCD).

To create a zone, follow these steps:

- In the **DCD**, go to **Menu** (left upper corner) > **Network Services** > **Cloud DNS**



- Click **Create > Primary Zone** in the **Public Zones tab** to open the **Create Primary Zone window**.
- Enter the following details in the **Create Primary Zone** window:
    - **Enabled/Disabled**: Set the status to either **Enabled** (Default) or Disabled.
    - **Name**: Enter an appropriate name for your DNS zone, **here ocp.ionos.yourdomain.de**
    - **Description (Optional)**: Enter an appropriate description for your DNS zone.
- Click **Create Zone** to create the DNS zone.

More detailed information about this step can be found in the IONOS CLOUD documentation.

6.4.5 Create DNS records

As we are going to install a High Available OpenShift Container Platform Cluster (see Chapter 2.3), we need DNS records for three Control Plane Nodes and two Compute Nodes.

Additionally, we will need DNS records for the Kubernetes API and the Application Ingress traffic.

To create DNS records in the DNS Zone we configured in the previous chapter:

- In the **DCD**, go to **Menu** > **Network Services** > **Cloud DNS** > **Public Zones** tab
- Select the appropriate zone in the **DNS ZONES** column to create records. Alternatively, click **Details & Records** in the **ACTIONS** column.
- Click **Create Record** in the Details & Records window.
- Enter the following details in the **Create Record** window:
    - **Enabled/Disabled**: Set it to Enabled (by default)
    - **Name**: Enter an appropriate name for the DNS record, see the list below
    - **TTL**: Enter an appropriate Time-To-Live (TTL) setting in seconds for your DNS record. Leave the default value at 3600 seconds.
    - **Type**: Select record types: A
    - **Content**: Enter the appropriate IPv4 address; see the list below
    - **Preview**: Ensure that the details of the record to be created are accurate.

FQDN	**IPv4 Address**
control-0.ocp.ionos.yourdomain.de	`10.7.200.xx`
control-1.ocp.ionos.yourdomain.de	`10.7.200.xx`
control-2.ocp.ionos.yourdomain.de	`10.7.200.xx`
compute-0.ocp.ionos.yourdomain.de	`10.7.200.xx`
compute-1.ocp.ionos.yourdomain.de	`10.7.200.xx`
api-int.ocp.ionos.yourdomain.de	`10.7.200.5` (internal Load-Balancer IP)
api.ocp.ionos.yourdomain.de	Public Load-Balancer IP from Chapter 4.4
*.apps.ocp.ionos.yourdomain.de	Public Load-Balancer IP from Chapter 4.4

More details about IONOS CLOUD-DNS and DNS record management can be found in the IONOS CLOUD documentation.

Note:
Copy and paste the nameservers of the new created zone to configure the domain at your Registrar. New DNS zones are currently not automatically registered.

6.4.6 Set the Availability Zone for the Dedicated Core Servers in DCD

The previous step created all the needed VMs within the IONOS CLOUD VDC. From an high availability perspective, it's important to distribute the VMs into different zones.
The IONOS CLOUD Region "Frankfurt am Main" (de/fra/2) we are using within this Deployment Guide has two Availability Zones (AZ) called "Zone_1" and "Zone_2". By default, the IONOS CLOUD VMs created previously are configured with zone "Auto".
To improve reliability and to configure OpenShift Container Platform zone awareness in a later step, it's recommended to distribute the VMs between those two available zones manually.
Please login to the IONOS CLOUD Data Center Designer (DCD) as described in Chapter 4.1 and check with the IONOS CLOUD documentation about how to configure a specific availability zone for every VM.

Remember to "**Provision Changes**" and note down the VM / AZ placement.

6.4.7 Create the Agent-based Installer ISO image

- Log in to the "Management" VM, as described in Chapter 6.3.1 in more detail.
- Create directories to store and backup the configuration files:

```
None
# mkdir $HOME/ocp-install $HOME/install-backup
```

- Create the `install-config.yaml` file by running the following command:

```
None
# cat << EOF > $HOME/ocp-install/install-config.yaml
apiVersion: v1
baseDomain: ionos.yourdomain.de
compute:
- architecture: amd64
  hyperthreading: Enabled
  name: worker
  replicas: 2
controlPlane:
  architecture: amd64
  hyperthreading: Enabled
  name: master
  replicas: 3
metadata:
  name: ocp
networking:
```

```
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  machineNetwork:
  - cidr: 10.7.200.0/23    <<<<<---------------- !!!!!!!!
  networkType: OVNKubernetes
  serviceNetwork:
  - 172.30.0.0/16
platform:
  external:
    platformName: custom-provider-name
sshKey: 'SSH pub key'   <<<<<---------------------------- !!!!!!!!!!!!!!!!!
pullSecret: 'your Red Hat Pull Secret' <<<<<------------- !!!!!!!!!!!!!!!!!
EOF
```

- Edit the `install-config.yaml` file:
  - Change the `sshKey` with the SSH Public Key you prepared in Chapter 3.5.2 named “ocp-admin”
  - Change the `pullSecret` with your Red Hat Pull Secret obtained within Chapter 3.5.1
  - Change the machineNetwork CIDR to your internal network address, see Chapter 6.3.1

- Create the `agent-config.yaml` file by running the following command:

```
None
# cat << EOF > $HOME/ocp-install/agent-config.yaml
apiVersion: v1beta1
kind: AgentConfig
metadata:
  name: ocp  <<<<<------------ !!!!!!!!!! cluster id configured in the DNS
# All fields are optional
rendezvousIP: (replace with IP of control-0 VM, i.e. 10.7.200.12)
bootArtifactsBaseURL: (replace with internal IP of Management VM, i.e.
http://10.7.200.11)
hosts:
  - hostname: control-0
    role: master
    interfaces:
      - macAddress: 02:01:e1:40:fe:50
        name: ens6
    networkConfig:
```

```
    interfaces:
      - name: ens6
        type: ethernet
        state: up
        mac-address: 02:01:e1:40:fe:50
        ipv4:
          enabled: true
          dhcp: false
          address:
            - ip: 10.7.200.12
              prefix-length: 23
    dns-resolver:
      config:
        server:
          - 212.227.123.16
          - 212.227.123.17
    routes:
      config:
        - destination: 0.0.0.0/0
          next-hop-address: 10.7.220.1
          next-hop-interface: ens6
          table-id: 254
- hostname: control-1
  role: master
  interfaces:
    - macAddress: 02:01:e1:40:fe:51
      name: ens6
  networkConfig:
    interfaces:
      - name: ens6
        type: ethernet
        state: up
        mac-address: 02:01:e1:40:fe:51
        ipv4:
          enabled: true
          dhcp: false
          address:
            - ip: 10.7.200.13
              prefix-length: 23
    dns-resolver:
      config:
        server:
          - 212.227.123.16
          - 212.227.123.17
    routes:
```

```
      config:
        - destination: 0.0.0.0/0
          next-hop-address: 10.7.200.1
          next-hop-interface: ens6
          table-id: 254
- hostname: control-2
  role: master
  interfaces:
    - macAddress: 02:01:e1:40:fe:52
      name: ens6
  networkConfig:
    interfaces:
      - name: ens6
        type: ethernet
        state: up
        mac-address: 02:01:e1:40:fe:52
        ipv4:
          enabled: true
          dhcp: false
          address:
            - ip: 10.7.200.14
              prefix-length: 23
    dns-resolver:
      config:
        server:
          - 212.227.123.16
          - 212.227.123.17
    routes:
      config:
        - destination: 0.0.0.0/0
          next-hop-address: 10.7.200.1
          next-hop-interface: ens6
          table-id: 254
- hostname: compute-0
  role: worker
  interfaces:
    - macAddress: 02:01:e1:50:fe:50
      name: ens6
  networkConfig:
    interfaces:
      - name: ens6
        type: ethernet
        state: up
        mac-address: 02:01:e1:50:fe:50
        ipv4:
```

```
            enabled: true
            dhcp: false
            address:
              - ip: 10.7.200.15
                prefix-length: 23
      dns-resolver:
        config:
          server:
            - 212.227.123.16
            - 212.227.123.17
      routes:
        config:
          - destination: 0.0.0.0/0
            next-hop-address: 10.7.200.1
            next-hop-interface: ens6
            table-id: 254
  - hostname: compute-1
    role: worker
    interfaces:
      - macAddress: 02:01:e1:50:fe:51
        name: ens6
    networkConfig:
      interfaces:
        - name: ens6
          type: ethernet
          state: up
          mac-address: 02:01:e1:50:fe:51
          ipv4:
            enabled: true
            dhcp: false
            address:
              - ip: 10.7.200.16
                prefix-length: 23
      dns-resolver:
        config:
          server:
            - 212.227.123.16
            - 212.227.123.17
      routes:
        config:
          - destination: 0.0.0.0/0
            next-hop-address: 10.7.200.1
            next-hop-interface: ens6
            table-id: 254
EOF
```

- **Important notes**:
  - we are using exactly the same `mac-addresses` as in Terraform when creating the VMs in order, so we know which VM has which MAC address on the interface
  - For every host defined in the `AgentConfig` file, we are setting a static IP configuration: IP address to the same the IONOS CLOUD DHCP gave back in Chapter 6.4.2, setting DNS IP addresses to the 1&1 resolvers, setting the default route to the internal IP address of the IONOS CLOUD NAT Gateway configured earlier.
  - `rendezvousIP` to one of the Control Plane Nodes IPs
  - `bootArtifactsBaseURL` to the Apache Http Server running on the "Management" VM and later providing the Red Hat CoreOS rootfs-image

- Prepare additional OpenShift Manifests to be included in the installation

  At the point of writing this Deployment Guide, OpenShift Container Platform is not correctly enabling the SystemD `nodeip-configuration.service` after the installation. This will lead to a missing "node-ip" configuration file for the `Kubelet`. A reasonable workaround is to provide a custom `MachineConfig` file to the `openshift-installer`:

```
None
# mkdir $HOME/ocp-install/openshift
# cat << EOF >
$HOME/ocp-install/openshift/99-enable-nodeip-custom-config-master.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
  name: 99-enable-nodeip-custom-config-master
spec:
  config:
    ignition:
      version: 3.4.0
    systemd:
      units:
        - name: nodeip-configuration.service
          enabled: true
EOF

# cat << EOF >
$HOME/ocp-install/openshift/99-enable-nodeip-custom-config-worker.yaml
apiVersion: machineconfiguration.openshift.io/v1
```

```
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 99-enable-nodeip-custom-config-worker
spec:
  config:
    ignition:
      version: 3.4.0
    systemd:
      units:
        - name: nodeip-configuration.service
          enabled: true
EOF
```

- As the `openshift-install` tooling is “consuming” the configuration files, it is recommended to back them up:

```
None
# cp $HOME/ocp-install/*-config.yaml $HOME/install-backup/
```

- Create the **agent image** by running the following command:

```
None
# openshift-install --dir $HOME/ocp-install agent create image
INFO Configuration has 3 master replicas and 2 worker replicas
INFO The rendezvous host IP (node0 IP) is 10.7.220.12
INFO Extracting base ISO from release payload
INFO Verifying cached file
INFO Using cached Base ISO /root/.cache/agent/image_cache/coreos-x86_64.iso
INFO Consuming Extra Manifests from target directory
INFO Consuming Agent Config from target directory
INFO Consuming Install Config from target directory
INFO RootFS file created in: boot-artifacts. Upload it at
http://10.7.200.11/agent.x86_64-rootfs.img
INFO Generated minimal ISO at agent.x86_64.iso
INFO When using External oci platform, always make sure CCM manifests were added
in the openshift directory.
```

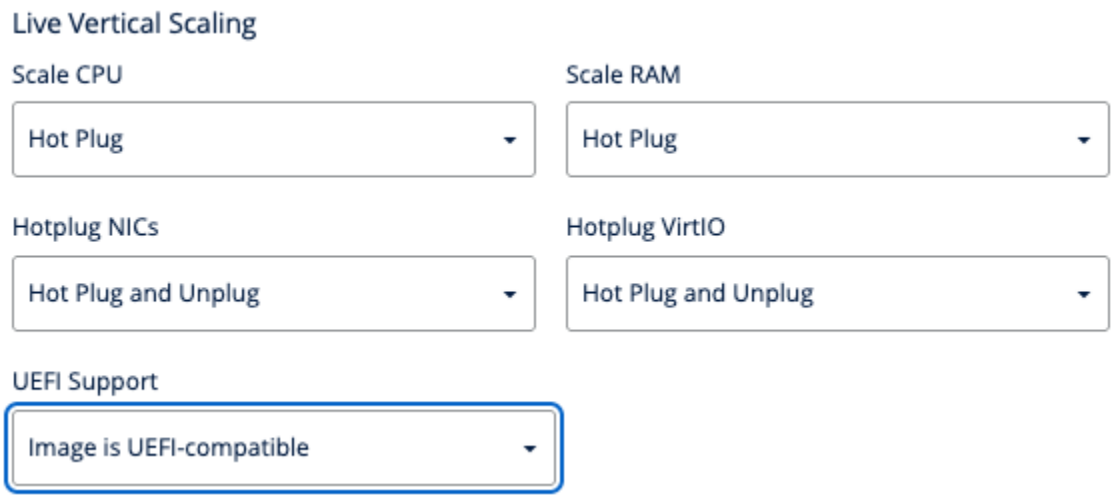
- Inspecting the $HOME/ocp-install directory, we can see the following files:

```
None
# tree -h
.
├── [ 115M]  agent.x86_64.iso
├── [   50]  auth
│   ├── [   23]  kubeadmin-password
│   └── [ 8.7K]  kubeconfig
├── [   37]  boot-artifacts
│   └── [ 1.1G]  agent.x86_64-rootfs.img
└── [   11]  rendezvousIP

2 directories, 5 files
```

- The "`agent.x86_64.iso`" file now contains the bootable image we will use to provision the VMs prepared previously with "Terraform".
  To have the ISO image available for provisioning, we need to upload the ISO image to the appropriate IONOS CLOUD data center via FTPS.
  As we are installing in the "**Germany/Frankfurt am Main**" data center, you need to use the following location: `ftps://ftp-fra-2.ionos.com/hdd-images`
  More details about uploading can be found in the IONOS CLOUD documentation.

- After the upload finished, set the vertical scaling flags like Scale CPU, Scale RAM to **Hot Plug**, Hotplug VirtIO to **Hot Plug and Unplug** and **UEFI** compatibility in the Manage Images and Snapshots window for that agent ISO image.



- The `agent.x86_64-rootfs.img` in the `boot-artifacts` directory contains the Red Hat CoreOS rootfs image, which is pulled by the init-ramdisk contained in the `agent.x86_64.iso` image.

As the IONOS CLOUD DHCP is not setting a default gateway nor DNS nameserver IPs for private networks, we need to have the rootfs image available within the private network.
For this reason, we already installed and enabled the Apache Http Server in Chapter 6.3.4.
Please copy the root-fs image to the webservers serving directory:

```
None
# cp $HOME/ocp-install/boot-artifacts/agent.x86_64-rootfs.img /var/www/html/
```

6.4.8 Attach “Minimal ISO” as CDROM drive and set boot order

During Chapter 6.4.2, we provisioned IONOS CLOUD Dedicated Core Servers with “empty” SSD disk drives. At this point, we couldn’t attach a CD-ROM drive for booting because the ISO Image for the Agent-based OpenShift Container Platform installation did not exist yet.

As we now have the ISO boot image (`agent.x86_64.iso`) prepared and uploaded to the IONOS CLOUD FTPS server, we can configure a CD-ROM device.

The Deployment Guide is currently missing a “Terraform” automation for this part, so we are using the Data Center Designer (DCD) at this stage once again:

- In the **DCD**, **select** one of the OpenShift VMs in your datacenter
- Navigate to the **Inspector** pane on the **right side** and select the **Storage tab**
- Click **Add CD-ROM** and **select** the `agent.x86_64.iso` image from your **own images**
- **Uncheck** the “**Boot from Device**” checkbox
- **Set** the already **existing SSD** device (named i.e. control-1-storage) **as Boot Device**
- **Repeat** the same **for all** OpenShift VMs

As described in the Agent-based Installer Workflow (see Chapter 6.1.3), this boot device settings helps to eliminate the need to change the boot order or remove the CD-ROM device after the Red Hat CoreOS image is written to disk and the VM is automatically rebooted for the second phase of the installation.

Press **Provision Changes** in the **DCD** to reboot all OpenShift VMs with the Minimal ISO image and to start the OpenShift deployment.

6.4.9 Monitor the OpenShift Container Platform Installation progress

With the previous step in rebooting the IONOS CLOUD Dedicated Core Server VMs, the Minimal ISO will be booted and the OpenShift Container Platform installation will start.

To monitor the installation progress, run the following commands and monitor the output on the “Management” Server:

- Monitoring the first part of the installation:

```
None
# openshift-install agent wait-for bootstrap-complete --log-level info
INFO Waiting for cluster install to initialize. Sleeping for 30 seconds
INFO Cluster is not ready for install. Check validations
WARNING Cluster validation: The cluster has hosts that are not ready to install.
INFO Cluster is ready for install
INFO Cluster validation: All hosts in the cluster are ready to install.
INFO Host control-1 validation: Host has connectivity to the majority of hosts in
the cluster
INFO Host control-0 validation: Host has connectivity to the majority of hosts in
the cluster
INFO Host control-2 validation: Host has connectivity to the majority of hosts in
the cluster
INFO Host compute-3 validation: Host has connectivity to the majority of hosts in
the cluster
INFO Host compute-0 validation: Host has connectivity to the majority of hosts in
the cluster
INFO Host compute-1 validation: Host has connectivity to the majority of hosts in
the cluster
INFO Host compute-2 validation: Host has connectivity to the majority of hosts in
the cluster
INFO Host control-2: updated status from insufficient to known (Host is ready to
be installed)
INFO Preparing cluster for installation
INFO Cluster installation in progress
INFO Host control-1: updated status from preparing-successful to installing
(Installation is in progress)
INFO Host: control-0, reached installation stage Writing image to disk: 39%
INFO Host: compute-0, reached installation stage Writing image to disk: 75%
INFO Host: compute-0, reached installation stage Writing image to disk: 100%
INFO Host: compute-0, reached installation stage Waiting for control plane
INFO Bootstrap Kube API Initialized
INFO Host: compute-2, reached installation stage Rebooting
INFO Host: compute-3, reached installation stage Rebooting
INFO Host: compute-0, reached installation stage Rebooting
INFO Host: control-0, reached installation stage Waiting for bootkube: waiting for
ETCD bootstrap to be complete
INFO Host: compute-3, reached installation stage Joined
INFO Bootstrap configMap status is complete
INFO Bootstrap is complete
```

```
INFO cluster bootstrap is complete
#
```

- Once the Cluster bootstrap is complete, the command exits and we can monitor the installation process to complete:

```
None
# openshift-install agent wait-for install-complete --log-level info
INFO Bootstrap Kube API Initialized
INFO Bootstrap configMap status is complete
INFO Bootstrap is complete
INFO cluster bootstrap is complete
INFO Cluster is installed
INFO Install complete!
INFO To access the cluster as the system:admin user when using 'oc', run
INFO     export KUBECONFIG=/root/ocp-install/auth/kubeconfig
INFO Access the OpenShift web-console here:
https://console-openshift-console.apps.ocp.ionos.yourdomain.de
INFO Login to the console with user: "kubeadmin", and password:
"xxxx-xxxxx-xxxxx-xxxxx"
```

At this point OpenShift Container Platform is successfully installed on IONOS CLOUD.

6.4.10 Post-install configuration

In a previous step (see Chapter 6.4.6) we distributed the OpenShift Container Platform Nodes to different Availibility Zones (AZ) in the IONOS CLOUD datacenter location.

However, OpenShift Container Platfom can not detect the zone location of the individual Nodes, as the VMs are not exposing this and we currently have no deeper integration into the IONOS CLOUD to check for metadata.

To add scheduling and high-availability awareness to OpenShift Container Platform, please use the following command to set the AZ accordingly:

```
None
# oc label node control-0 topology.kubernetes.io/zone=ZONE_1
```

```
# oc label node control-1 topology.kubernetes.io/zone=ZONE_2
# oc label node control-2 topology.kubernetes.io/zone=ZONE_1
# oc label node compute-0 topology.kubernetes.io/zone=ZONE_1
# oc label node compute-1 topology.kubernetes.io/zone=ZONE_2
```

7 Day 2 operations

7.1 Configure persistent storage

To use Block Storage volumes with OpenShift Container Platform, we need to provision and configure the IONOS CLOUD Blockstorage CSI driver version >= 0.6.0.
This Container Storage Interface (CSI) driver plugin communicates with the IONOS CLOUD API to manage storage. The visibility and permissions it has depend on the authentication token it is given.

7.1.1 Prerequisites

- Helm 3+ available
- IONOS CLOUD API Token
    - Please review Chapter 4.2 about how to create IONOS CLOUD API tokens. It is recommended creating a dedicated token for the CSI driver, potentially using a higher TTL as otherwise we need to change the token frequently

7.1.2 Create a secret with the IONOS CLOUD API Token for the CSI driver

The secret needs to be named csi-secret and the key needs to be named token:

```
None
$ oc -n kube-system create secret generic csi-secret --from-literal
token=<your-token>
```

7.1.3 Create IONOS CLOUD CSI Node Server config file on all Nodes

The CSI node server expects the file `/etc/ie-csi/cfg.json` to exist on every VM. The file must contain the datacenter ID of the VM in the following format:

```
None
{"datacenter-id": "<DATACENTER_ID>"}
```

Use `base64` to encode the above string and insert it in the `MachineConfig` manifests for OpenShift Container Platform Worker Nodes:

```
None
$ cat 99-ionos-csi-config-worker.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  name: 99-ionos-csi-config-worker
```

```
  labels:
    machineconfiguration.openshift.io/role: worker
spec:
  config:
    ignition:
      version: 3.4.0
    storage:
      files:
        - path: /etc/ie-csi/cfg.json
          mode: 0644
          contents:
            source:
data:text/plain;charset=utf-8;base64,replace-me-with-BASE64-encoded-string
```

Apply the MachineConfig:

```
None
$ oc apply -f 99-ionos-csi-config-worker.yaml
```

Once the Machine Config Pools are ready again, deploy the IONOS CLOUD CSI driver with helm:

```
None
$ helm install -n kube-system ionoscloud-blockstorage-csi-driver \
    oci://ghcr.io/ionos-cloud/helm-charts/ionoscloud-blockstorage-csi-driver \
    --set tokenSecretName=csi-secret --set init.selinux.enabled=true
```

7.2 Configuring Certificates for Ingress and API

7.2.1 Replacing the default Ingress Certificate

Applications are usually exposed at `<route_name>.apps.<cluster_name>.<base_domain>`. The `<cluster_name>` and `<base_domain>` come from the installation config file. `<route_name>` is the host field of the route, if specified, or the route name. For example, `hello-openshift-default.apps.username.devcluster.openshift.com`. `hello-openshift` is the name of the route and the route is in the default namespace.

You might want clients to access the applications without the need to distribute the cluster-managed CA certificates to the clients. The administrator must set a custom default certificate when serving application content.

By default, OpenShift Container Platform uses the Ingress Operator to create an internal CA and issue a wildcard certificate that is valid for applications under the `.apps` sub-domain. Both the web console and CLI use this certificate as well.

The internal infrastructure CA certificates are self-signed. While this process might be perceived as bad practice by some security or PKI teams, any risk here is minimal. The only clients that implicitly trust these certificates are other components within the cluster. Replacing the default wildcard certificate with one that is issued by a public CA already included in the CA bundle as provided by the container userspace allows external clients to connect securely to applications running under the .apps sub-domain.

To replace the default ingress certificate for all applications under the `.apps` subdomain, please follow the detailed steps and prerequisites outlined in the OpenShift Container Platform documentation.

If not already specified during the initial OpenShift Container Platform installation, please remember to also update the custom Certificate Authority (CA) to the “CA Bundle”. Fore more details, check the OpenShift Container Platform documentation

7.2.2 Replacing the default API Server Certificate

The API server is accessible by clients external to the cluster at `api.<cluster_name>.<base_domain>`. You might want clients to access the API server at a different hostname or without the need to distribute the cluster-managed certificate authority (CA) certificates to the clients.

The default API server certificate is issued by an internal OpenShift Container Platform cluster CA. Clients outside of the cluster will not be able to verify the API server’s certificate by default. This certificate can be replaced by one that is issued by a CA that clients trust.

The user-provided certificates must be provided in a `kubernetes.io/tls` type `Secret` in the `openshift-config` namespace. Update the API server cluster configuration, the `apiserver/cluster` resource, to enable the use of the user-provided certificate.

To replace the default API Server certificate, please follow the detailed steps and prerequisites outlined in the OpenShift Container Platform documentation.

7.3 Configure OpenShift Image Registry

OpenShift Container Platform can build images from your source code, deploy them, and manage their lifecycle. It provides an internal, integrated container image registry that can be deployed in your OpenShift Container Platform environment to locally manage images.

Due to the fact that we are using "`Platform: External`" for the OpenShift Container Platform installation, there is no default integration with the specific storage offerings on IONOS CLOUD.

For this reason, the OpenShift Image Registry Operator bootstraps itself as `Removed`, which means that there is no OpenShift internal Image Registry configured at all.

A scaled OpenShift image registry involves running multiple replicas of the registry to improve performance, reliability, and availability.
By scaling the OpenShift image registry, you can ensure that your cluster can efficiently manage and distribute container images, supporting large-scale deployments and improving overall system reliability.

According to the recommendations for the Registry storage in the documentation, it is needed to use object storage for the scaled OpenShift Image Registry. Please refer to Chapter 3.2.4 to learn more about IONOS CLOUD Object Storage.

The OpenShift Container Platform documentation includes a detailed description about how to configure and set up the OpenShift Image Registry, please follow the steps in the Chapter "Configuring the registry for bare metal".

To set up and create an object storage bucket on IONOS CLOUD, please follow the steps outlined in the IONOS CLOUD documentation.

After creating the IONOS CLOUD Object Storage bucket, you need to create an Object Storage Key as outlined in the IONOS CLOUD documentation.

Once we have the "Access Key" and the "Secret Key", we are going to create the needed OpenShift Secret in the `openshift-image-registry` namespace:

```
None
$ oc create secret generic image-registry-private-configuration-user
--from-literal=REGISTRY_STORAGE_S3_ACCESSKEY=your-key-here \
--from-literal=REGISTRY_STORAGE_S3_SECRETKEY=your-key-here \
--namespace openshift-image-registry
```

Now we need to change the OpenShift Image Registry configuration by editing with `oc edit`:

```
None
$ oc edit config.imageregistry.operator.openshift.io

Example Manifest, change spec.managementState, spec.replicas and spec.storage:

apiVersion: imageregistry.operator.openshift.io/v1
kind: Config
metadata:
  name: cluster
spec:
  logLevel: Normal
  managementState: Managed
  operatorLogLevel: Normal
  proxy: {}
  replicas: 2
  rolloutStrategy: RollingUpdate
  storage:
    s3:
      bucket: openshift-registry
      region: eu-central-3
      regionEndpoint: https://s3.eu-central-3.ionoscloud.com
  unsupportedConfigOverrides: null
```

7.4 Configure OpenShift Monitoring

When configuring OpenShift monitoring, using persistent storage is highly recommended. Persistent storage ensures that your metrics and alerting data are protected from loss when pods are restarted or recreated. This is crucial for maintaining historical data and ensuring continuity in monitoring and alerting systems.

To implement persistent storage, you need to configure Persistent Volume Claims (PVCs) for monitoring components and ensure sufficient local storage is available.

OpenShift Container Platform Monitoring is using Prometheus databases for storing metrics and alerts. The recommended storage technology for Prometheus is **block storage**.

Within the OpenShift Container Platform documentation, you will find some numbers regarding the required storage capacity for different Cluster sizes. Keep in mind that by default, Prometheus retains metrics for 15 days. You can modify the retention time for the Prometheus instance to change when the data is deleted. You can also set the maximum amount of disk

space the retained metrics data uses. More details about modifying retention time and size is available in the documentation as well.

The main steps to configure persistent storage for OpenShift Container Platform Monitoring are:

- **Edit ConfigMap**:
  - Modify `cluster-monitoring-config` in the `openshift-monitoring` namespace.
- **Add PVC Configuration**:
  - Specify `volumeClaimTemplate` for each component (e.g., Prometheus, Alertmanager) with storage class and size.
- **Ensure PVs Are Available**:
  - Have sufficient PVs for each replica.
- **Verify Configuration**:
  - Check that pods use the specified storage after applying changes.

Please refer to the OpenShift Container Platform documentation for detailed instructions.

7.5 Scaling the OpenShift Cluster

OpenShift Container Platform is a highly scalable solution and Red Hat is publishing some numbers about the tested Cluster maximums in the documentation.

However, the environment, application workload and infrastructure are never the same and it's hard to compare with any general numbers. Scaling up, preparing for more workload and keeping a decent end-customer performance needs proper planning and testing.

The OpenShift Container Platform documentation provides a whole chapter about "Scalability and performance", covering a broad number of different topics. It is recommended to get familiar with those topics already in the planning and design phase.

As a general guidance, it is very important to keep the Control Plane Node size inline with the overall Cluster growth. The node sizing varies depending on the number of Worker Nodes and object counts in the cluster. It also depends on whether the objects are actively being created on the cluster. The "Control Plane Node sizing" chapter in the documentation has some numbers.

7.5.1 Adding Worker Nodes to the Cluster

Starting with OpenShift Container Platform 4.17, you can add Worker Nodes by using the `oc adm node-image` command to generate an ISO image, which can then be used to boot one or more nodes in your target cluster.

With this platform-agnostic approach, you can add one or more Nodes at a time while customizing each Node with more complex configurations, such as static network configuration. Any required configurations that are not specified during ISO generation are retrieved from the target cluster and applied to the new nodes.

Preflight validation checks are also performed when booting the ISO image to inform you of failure-causing issues before you attempt to boot each node.

Before running the `oc adm node-image create` command to generate the ISO image suitable for your OpenShift Container Platform Cluster, you need to prepare the VMs on IONOS CLOUD. Please refer to Chaper 6.4 about how to create the VM instances.

The detailed steps needed to create the ISO image for one or more Nodes is outlined in the OpenShift Container Platform documentation.

7.6 OpenShift Container Platform updates

7.6.1 Understanding OpenShift updates

OpenShift Container Platform updates are managed through the **Cluster Version Operator (CVO)** and the **OpenShift Update Service (OSUS)**.
The OSUS provides a graph of update possibilities based on release images, ensuring compatibility and safety. When an update is requested, the CVO retrieves the target release image and applies changes to the cluster.
The **Machine Config Operator (MCO)** handles node updates by cordoning nodes, applying new configurations, and rebooting them. Updates are typically rolled out in stages, and only upgrading to newer versions is supported. The OSUS continuously checks for available updates and notifies administrators when new versions are ready.

More details about OpenShift Container Platform updates as well as a collection of frequently asked questions can be found in the documentation.

Attention:
Before updating the OpenShift Cluster to any new z-stream or minor release, it is highly recommended to check the OpenShift Container Release Notes for the appropriate release and to work through the "Preparing to update a cluster" chapter in the documentation.

7.6.2 Updating to the next OpenShift z-stream maintenance release

To update OpenShift Container Platform to the latest z-stream release (i.e. 4.17.1 to 4.17.3) using the command line, follow these steps:

- **Install the Correct CLI Version**: Ensure you have the OpenShift CLI (`oc`) version that matches your target update version.

- **Log In with Admin Privileges**: Log in to the cluster as a user with `cluster-admin` privileges.
- **Pause Machine Health Checks**: Pause all `MachineHealthCheck` resources to prevent interference during the update.
- **Check Available Updates**: Use `oc adm upgrade` to view available updates and note the desired version.
- **Apply the Update**: Use `oc adm upgrade --to-latest=true` to update to the latest version or specify a version with `--to <version>`.
- **Monitor the Update**: Use `oc get clusterversion` and `oc get nodes` to monitor the update progress until it completes. You can also use `oc adm upgrade` for monitoring.

More details about the procedure can be found in the OpenShift Container Platform documentation.

7.6.3 Upgrading to the next OpenShift minor release

Important:
When updating to a new OpenShift minor release (i.e. from 4.17 to 4.18), it is especially important to review and check the OpenShift Container Release Notes for the appropriate release. Consult the "Notable technical changes" and "Deprecated and removed features" sections carefully for the particular release.!

To update OpenShift Container Platform to a more recent minor release (i.e. 4.17.20 to 4.18.5) using the command line, follow these steps:

- **Install the Correct CLI Version**: Ensure you have the OpenShift CLI (`oc`) version that matches your target update version.
- **Log In with Admin Privileges**: Log in to the cluster as a user with `cluster-admin` privileges.
- **Pause Machine Health Checks**: Pause all `MachineHealthCheck` resources to prevent interference during the update.
- **Set the appropriate update channel:**
  Use `oc adm upgrade channel stable-4.18` to set the "stable-4.18" channel
- **Check Available Updates**: Use `oc adm upgrade` to view available updates and note the desired version.
- **Apply the Update**: Use `oc adm upgrade --to-latest=true` to update to the latest version or specify a version with `--to <version>`.
- **Monitor the Update**: Use `oc get clusterversion` and `oc get nodes` to monitor the update progress until it completes. You can also use `oc adm upgrade` for monitoring.

More details about the procedure can be found in the OpenShift Container Platform documentation.

8 Troubleshooting and Support

8.1 Gathering log data from a failed Agent-based installation

To gather log data about a failed Agent-based installation and to provide that for a support case, please refer to the OpenShift Container Platform documentation.

8.2 General OpenShift Container Platform Troubleshooting

The OpenShift Container Platform documentation includes a whole chapter about troubleshooting in different areas. Please consult the documentation to get started.