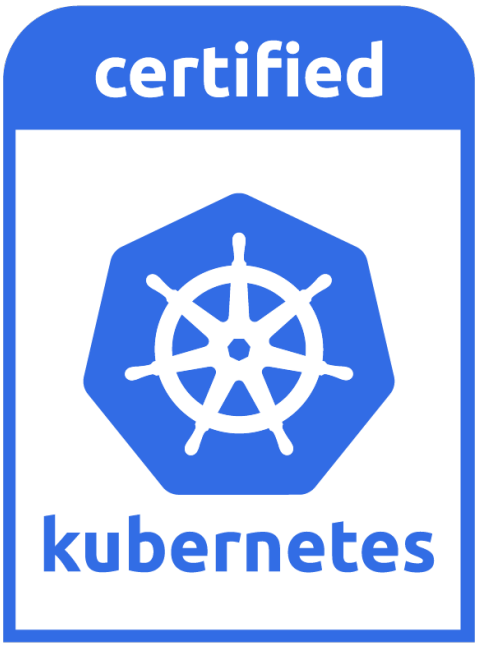
Whitepaper

Kubernetes Platform

Catalyst Cloud



Version 2.5

11 June 2020

catalyst cloud

Table of contents

1 Introduction

Catalyst Cloud's Kubernetes platform makes it easy to deploy, manage, and scale Kubernetes clusters to run containerised applications on the cloud. The platform is fully managed and removes the need for specialist knowledge, allowing developers to focus on applications instead of wrangling with infrastructure.

The Cloud Native Computing Foundation (CNCF) Certified Kubernetes Conformance Program guarantees application portability and assures customers that Kubernetes APIs work as designed. Catalyst Cloud is the first in New Zealand to provide a CNCF certified Kubernetes platform service onshore.

The certification process puts Catalyst Cloud among the front-runners of companies worldwide developing emerging standards for cloud computing and providing a world-class solution to the New Zealand market.

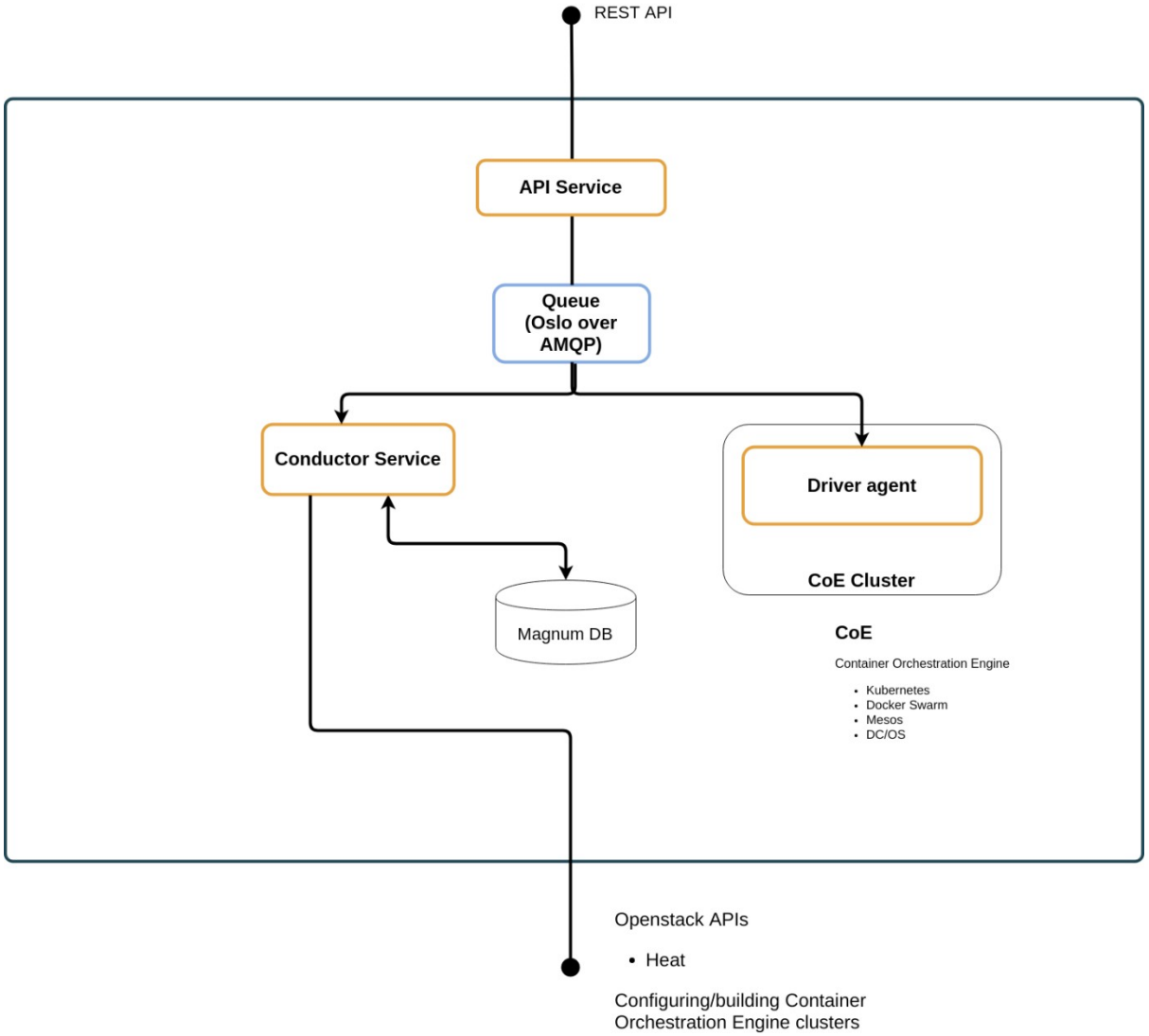
2 Container Infrastructure Management Service

The Container Infrastructure Management API allows customers to provision clusters based on certified cluster templates (internally called bay models). The example below illustrates a sample request to create a Kubernetes cluster with three masters and three worker nodes:


```
{
    "name":"k8s",
    "discovery_url":null,
    "master_count":3,
    "cluster_template_id":"0562d357-8641-4759-8fed-8173f02c9633",
    "node_count":3,
    "create_timeout":60,
    "keypair":"my_keypair",
    "master_flavor_id":null,
    "labels":{
    },
    "flavor_id":null
}
```


2.1 Logical architecture

The following diagram depicts the micro-services belonging to the Container Infrastructure Service in OpenStack and their interactions.



The functions performed by service components are:

- **API Service**: Accepts and responds to API calls, does some basic request sanitisation and then inserts the request in the work queue.
- **Conductor Service**: Processes work requests by using the appropriate Container Orchestration Engine driver to create the cluster using the Cloud Orchestration Service.
- **Driver Agent**: Agent running in the provisioned compute instances that communicates back with the Cloud Orchestration and Container Infrastructure Management control plane for deployment, configuration, and/or monitoring purposes.

2.2 Cluster templates

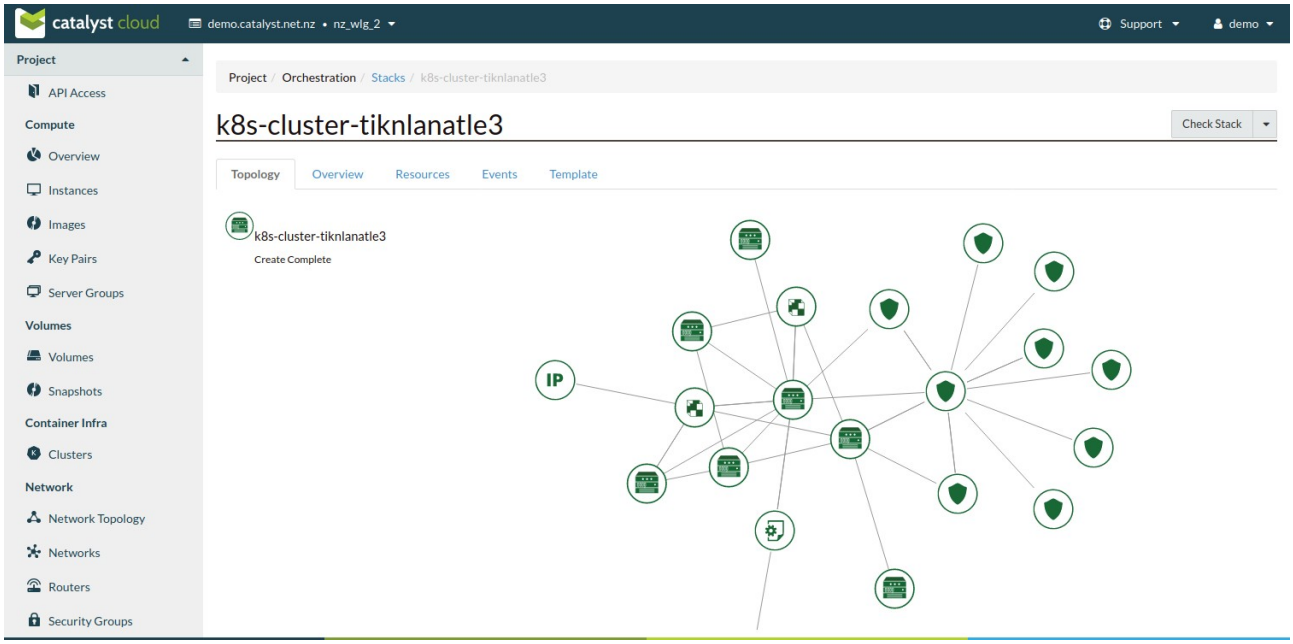
Cluster templates describe all the resources required to run Kubernetes on the cloud, such as compute instances, block volumes, software configuration, and security groups. The templates provided by Catalyst Cloud are CNCF certified, providing assurance to customers that Kubernetes works as designed and that applications can be easily migrated to/from other CNCF certified platforms.

2.3 Cluster deployment process

When an API call is submitted to the Container Infrastructure Management Service to provision a new cluster, the request is stored in a message queue and picked up asynchronously by the Conductor, a back-end process that fulfills the request. The Conductor replaces variables in the cluster template with the inputs provided by the user and calls the Cloud Orchestration API to deploy the cluster.

The Cloud Orchestration service enables the deployment of applications using a simple template language that describes the required resources and their relationship. The service manages the complete life-cycle of application stacks. When a new version of a template is applied, the service orchestrates the required changes to the infrastructure in the appropriate order.

When the API call from the Container Infrastructure Management Service is received, the Cloud Orchestration service produces a dependency graph (as seen on the screenshot below) with all the resources required to build the Kubernetes cluster. It traverses this graph, submitting API requests to the required cloud services like the Network, Compute, and Block Storage.



All the resources created by the Cloud Orchestration service and the events associated with them can be seen via the dashboard or APIs, providing complete transparency to users on what is happening behind the scenes. The following screenshot is an example of the audit trail available for all actions performed per user request or automated (such as auto-healing or automatic updates).

Stack Resource	Resource	Time Since Event	Status	Status Reason
k8s-cluster-tiknlanatle3	6d5270f6-6362-4063-898c-b5a6ecf21294	3 months	Create Complete	Stack CREATE completed successfully
kube_minions	1b1c5cb3-d04e-4e9c-b96d-10b5d9d3f2c0	3 months	Create Complete	state changed
kube_cluster_deploy	83e8634e-baf9-433f-97e4-cbf235f25f1f	3 months	Create Complete	state changed
kube_cluster_deploy	83e8634e-baf9-433f-97e4-cbf235f25f1f	3 months	Signal In Progress	Signal: deployment 83e8634e-baf9-433f-97e4-cbf235f25f1f succeeded
api_address_floating_switch	555a2e6c-7cbe-47cb-8922-d645caf33193	3 months	Create Complete	state changed
api_address_floating_switch	k8s-cluster-tiknlanatle3-api_address_floating_switch-ifue2fw63lra	3 months	Create In Progress	state changed

3 Kubernetes Platform

3.1 Cluster configuration

Compute instances

The compute instances (master and worker nodes) of the Kubernetes cluster are provisioned using the Fedora CoreOS image.

Fedora CoreOS is a minimalist operating system for running containerised workloads securely and at scale. It is mainly composed of the Linux kernel, systemd, Docker Engine, and rpm-ostree (a package management system that can perform atomic upgrade/rollback operations).

The filesystem of the base operating system is mounted in read-only mode, preventing potential attackers from making changes to the host. In addition, another layer of defense is implemented with the Linux kernel security module (SELinux) in enforcing mode, in line with security in-depth principles.

Kubernetes binaries

Catalyst Cloud has developed a CI/CD pipeline to test and build container images with statically linked binaries for all Kubernetes components, ensuring they pass a comprehensive automated test harness which includes the CNCF certification tests. These open source images are published to the Docker Hub.

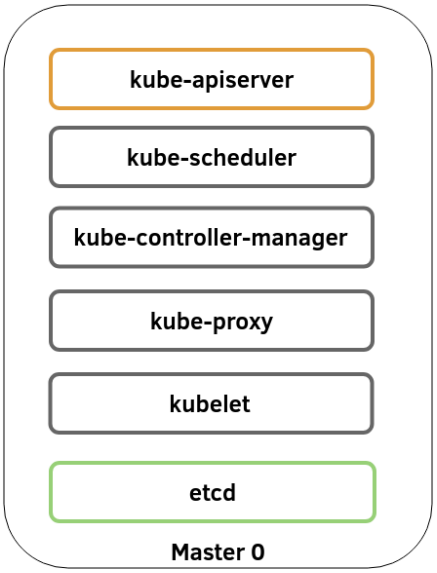
The Cloud Orchestration Service pulls these container images into the compute instances (master and worker nodes), while verifying they match the expected sha256 digest to ensure they have not been compromised in Docker Hub or in transit. Systemd is used to initialise and ensure the required containers are running on the master and worker nodes.

In summary, all software components required to run Kubernetes or supporting systems of the platform are containers pulled into the compute instances by the Driver Agent. These containers are stateless and follow the principle of immutable infrastructure (they are replaced with a new container version if changes or updates are required).

Master nodes

A Kubernetes cluster may have one or more master nodes. A production Kubernetes cluster will typically have three master nodes for high availability and resiliency. Master nodes are also referred to as the Kubernetes control plane.

Each master node has the following Kubernetes components deployed to it (as containers):



- **kube-apiserver**: binds to port 6443 TCP and responds to API requests
- **kube-scheduler**: responsible for scheduling Pods on the cluster
- **kube-controller-manager**: runs the core control loops in Kubernetes
- **kube-proxy**: a network proxy that allows network communication to Pods from network sessions inside or outside of your cluster
- **kubelet**: an agent that manages containers created by Kubernetes
- **etcd**: a distributed key-value data store used to store configuration data, metadata and state

Worker nodes

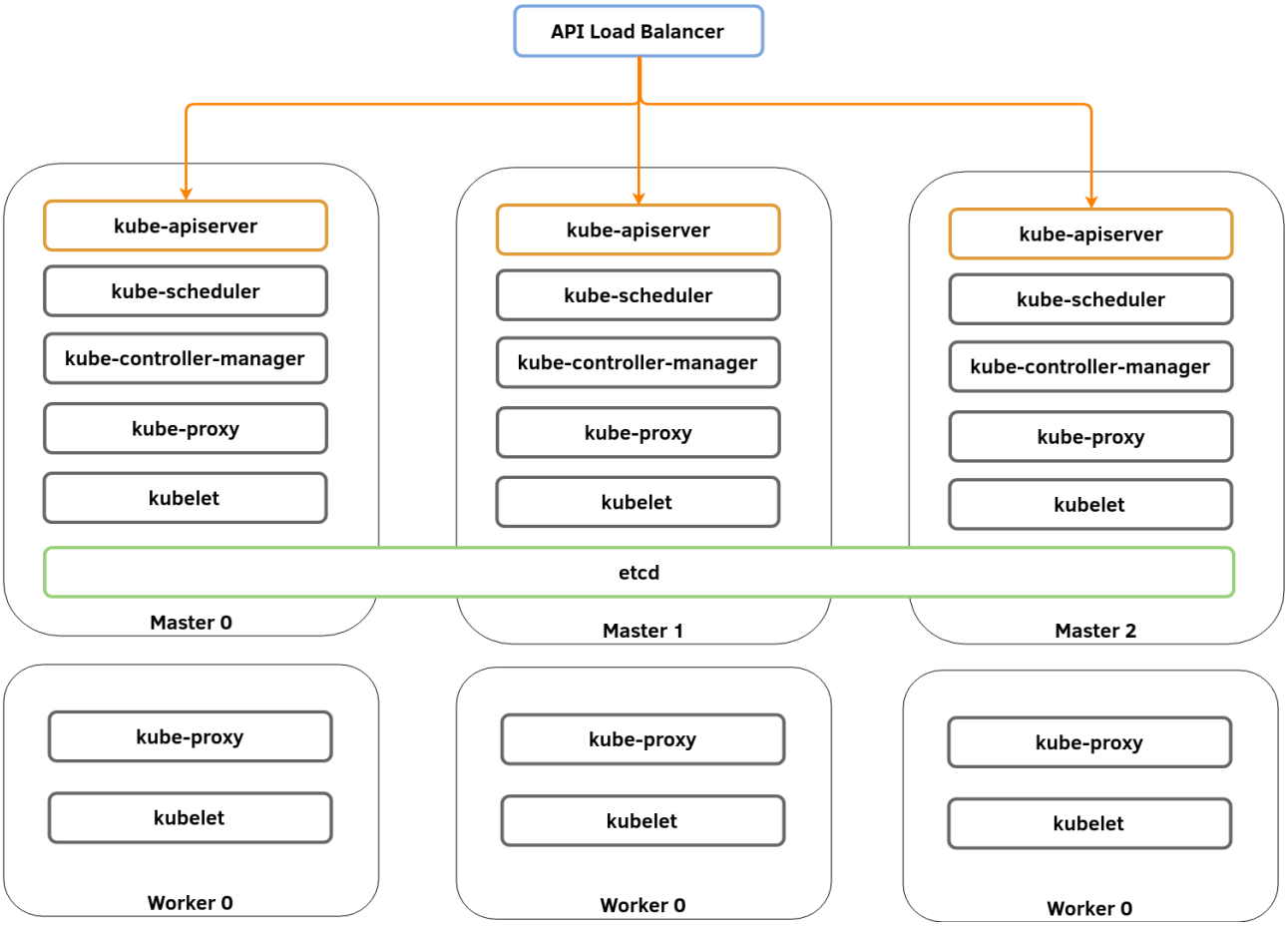
A Kubernetes cluster may have one or more worker nodes. A production Kubernetes cluster will typically have three or more worker nodes for high availability and resiliency.

Each worker node has the following Kubernetes components deployed to it (as containers):

- **kube-proxy**: a network proxy that allows network communication to Pods from network sessions inside or outside of your cluster
- **kubelet**: an agent that manages containers created by Kubernetes

Kubernetes cluster

A typical production Kubernetes cluster has at least three master and three worker nodes. A load balancer is used to distribute API requests to the "kube-api" process across master nodes. The load balancer implements a health check, ensuring API requests are only forwarded to healthy nodes.

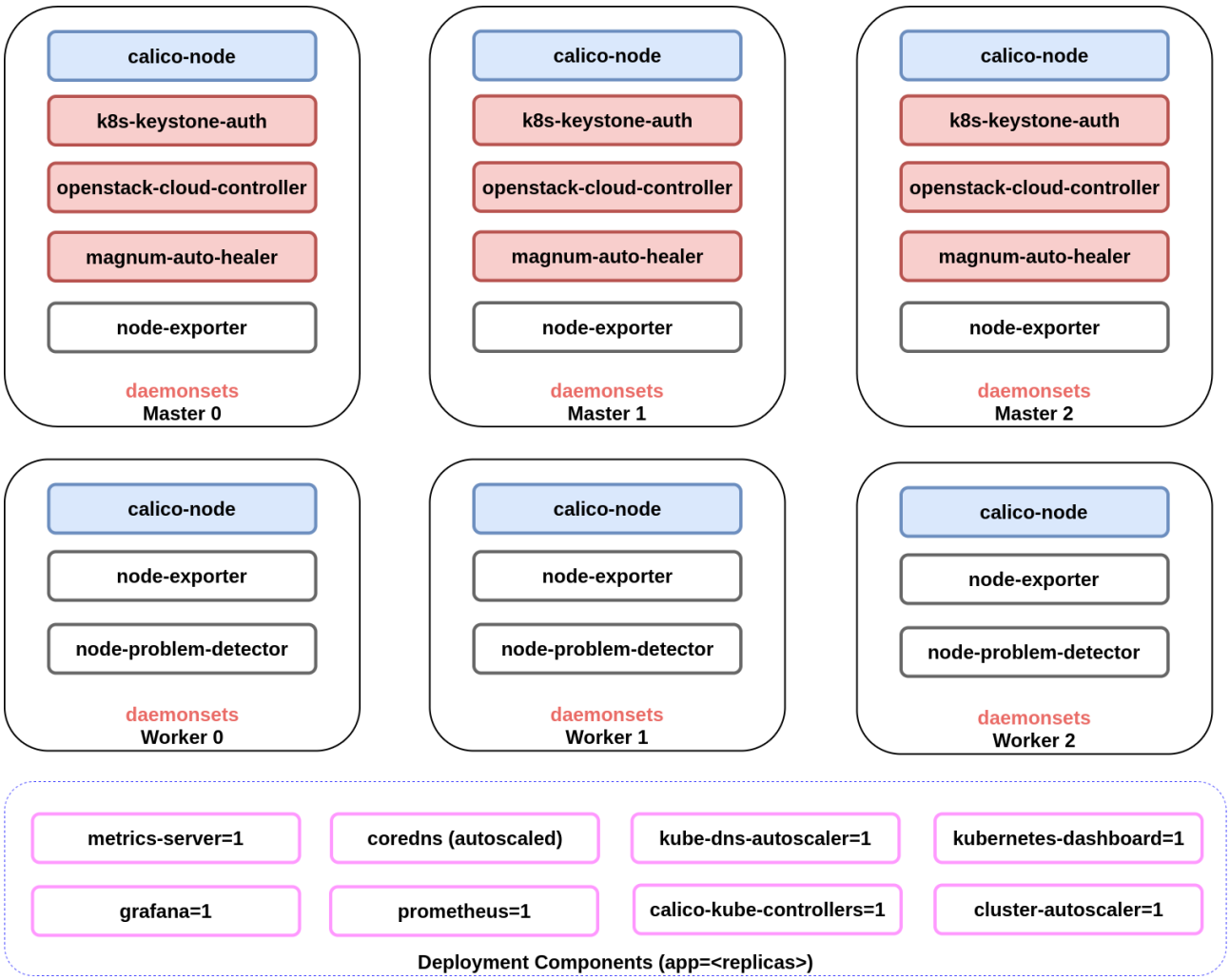


As a security best practice, the production templates on the Catalyst Cloud do not expose the API to the internet. Access to the API is limited by a security group to the CIDR of the private (RFC 1918) subnet that the Kubernetes cluster was created on. API traffic is encrypted in transit using TLS v3 (PKCS 1 SHA-256 RSA).

Components deployed into Kubernetes

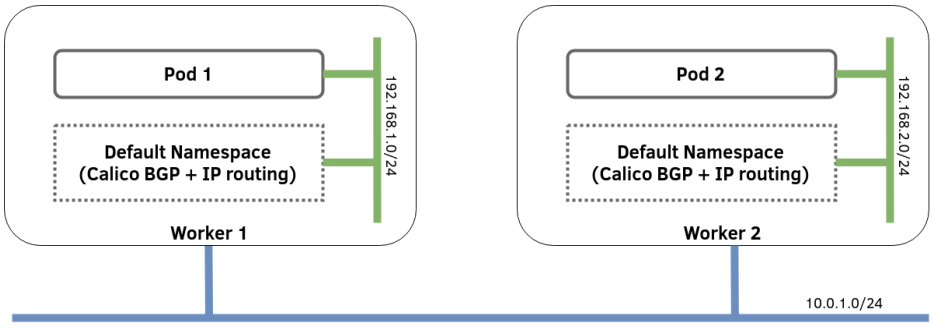
The following components are deployed into Kubernetes (under the “kube-system” namespace) to provide the functionality required by users and the integration expected with the Catalyst Cloud:

- **Calico:** the CNI backend used to provide network connectivity between Pods and enforce network policies (traffic filtering rules akin to security groups)
- **K8s Keystone Auth**: a webhook that forwards authentication requests to the Catalyst Cloud IAM service, enabling users to log in to Kubernetes using their cloud (fernet) token
- **OpenStack Cloud Controler**: a cloud provider controller manager that enables Kubernetes to interact with the Catalyst Cloud APIs (e.g. to allocate additional compute resources, persistent volumes, or load balancers)
- **Node Problem Detector**: monitors the health of individual nodes
- **Magnum Auto Healer**: a monitoring tool that checks the status of the Kubernetes cluster and reports its health state back to the Container Infrastructure Management service (for auto-healing purposes)
- **Node Exporter**: reads Kubernetes statistics and exports them to Prometheus for monitoring purposes
- **Prometheus**: monitoring system with a dimensional data model, flexible query language, efficient time-series database, and modern alerting approach
- **Cluster Autoscaler**: monitors the resources allocated by Pods and adds or removes worker nodes (in line with the minimum and maximum number defined)
- **Metrics Server**: collects resource metrics from Kubelets and exposes them in Kubernetes apiserver through Metrics API for use by Horizontal Pod Autoscaler and Vertical Pod Autoscaler

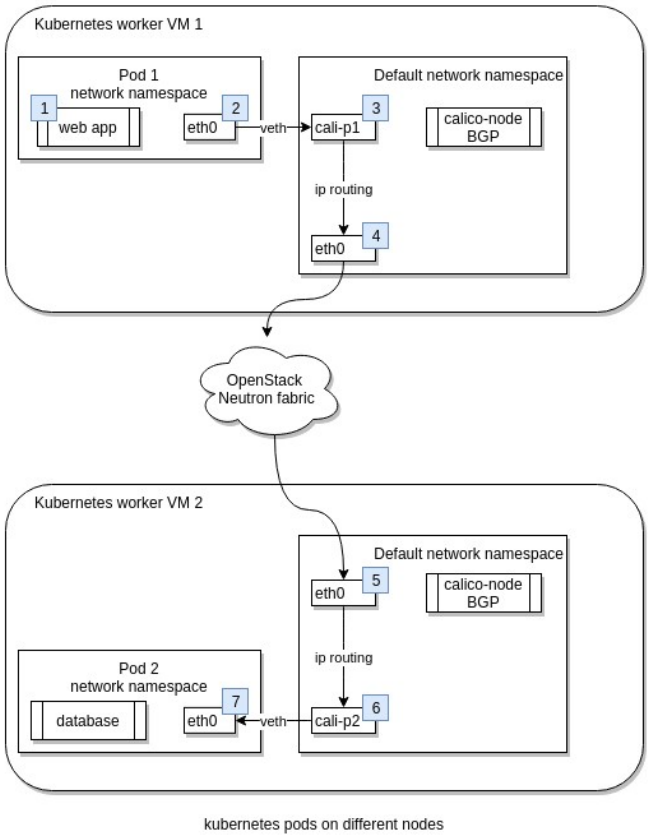


Network configuration

Calico is used as the CNI plugin to provide network connectivity between pods and traffic filtering. A private (RFC 1918) supernet block is reserved for the internal network within Kubernetes and /24 subnets carved out from it for each worker node, as illustrated in the diagram below:



Calico forms (using the Felix agent and the BIRD routing daemon) a BGP mesh between all nodes in the cluster, so that routing tables can be created and exchanged. In the example above, when Pod 1 wants to communicate with Pod 2, the following network path is followed:



kubernetes pods on different nodes

In addition to allowing pod-to-pod communication, Calico also implements and enforces network policies (traffic filtering for pods, akin to security groups for compute instances). More information on network policies can be found on the Catalyst Cloud documentation at: https://docs.catalystcloud.nz/kubernetes/network-policies.html

Authentication

There are three ways users can authenticate with their Kubernetes cluster:

- By using the root certificate generated at cluster creation time (this form of authentication is reserved for Kubernetes administrators and is not used for day-to-day tasks).
- By using the Catalyst Cloud credentials or an authentication token.
- By implementing their own webhook using their preferred IdP, LDAP, or Active Directory back-end for authentication.

More information on authentication can be found on the Catalyst Cloud documentation at: https://docs.catalystcloud.nz/kubernetes/access-control.html

Authorisation

Authorisation is controlled by Kubernetes RBAC configuration mappings. For convenience, there are three pre-defined cluster wide roles that can be granted using the Catalyst Cloud IAM service:

- **k8s_admin**: Allows users to perform CRUD operations to Magnum cluster and have full admin access to Kubernetes. Has access to all namespaces, including the admin namespace.
- **k8s_developer**: Allow users to perform CRUD operations to Kubernetes resources. The user has access to all namespaces, excluding the admin namespace.
- **k8s_viewer**: Only allows the user to perform READ operations in both Magnum and Kubernetes. Has access to all namespaces, excluding the admin namespace.

Users can modify or delete these pre-defined roles to suit the needs of their business. They can also create more granular roles that provide access to specific namespaces or resources in the cluster to a user or group of users.

Hardening

The platform is designed and developed from the ground up with very high levels of security in mind. The following list outlines some of the security principles and hardening best practices applied:

- **Immutable infrastructure:** Both the base operating system and the container file-systems are mounted as read-only. To make changes to the software, containers must be redeployed using a new container image that matches the sha256 digest specified by our engineers.
- **Minimum software:**
  - **Base operating system:** Only the minimum software required to run containers is present on the hosts.
  - **Containers:** All platform components are deployed in containers. Where possible the Kubernetes have only a single statically linked binary in them. We endevour to use Alpine Linux (a security-oriented, minimalist and lightweight Linux distribution) for other containers that require an operating system, such as those running applications written on interpreted languages (like Python).
- **Least privilege:** Application access on the base host and containers is limited to the information and resources necessary for the application to function.

- **Network filtering:** Cloud security groups (akin to firewalls, but entirely software-defined) are used to restrict the communication between components of the solution to the minimum required by Kubernetes.
- **Data encrypted in transit:** All cluster communication (between Kubernetes components, between Kubernetes and the Cloud IaaS services, and between the user and the Kubernetes APIs) is encrypted in transit using TLS v3 (PKCS 1 SHA-256 RSA). Kubernetes certificates can be easily rotated by users via a simple API call.
- **Data encrypted at rest:** Volumes used for the base operating system, Docker images, etcD, or persistent volumes created by users are encrypted at rest using an AES 256-bit cipher by default.
- **Security in depth:** Multiple layers are used to protect application workloads, including but not limited to network policies, containers, kernel security modules, hypervisors, and security groups.

3.2 Cluster management

Monitoring

The health state of each Kubernetes cluster is monitored by the Catalyst Cloud both from inside the cluster (using the "Magnum Auto Healer") and from the outside. If a master or worker node is not functioning as required, an auto-healing action is triggered by the Container Infrastructure Management service to repair the impacted node. More information about auto-healing can be found on the Catalyst Cloud documentation: https://docs.catalystcloud.nz/kubernetes/auto-healing.html

Security updates & upgrades

The upgrade API call enables users to easily upgrade the version of a running Kubernetes cluster with minimum or no impact on running applications. More information on how rolling upgrades work can be found on the Catalyst Cloud documentation: https://docs.catalystcloud.nz/kubernetes/rolling-upgrade.html

Security updates are provided by the Catalyst Cloud in the form of a new template version (with a patch version or date suffix increment). These updates may contain security patches to the base operating system (Fedora CoreOS), the Docker Engine, or the container images (application binaries) used to deploy Kubernetes.

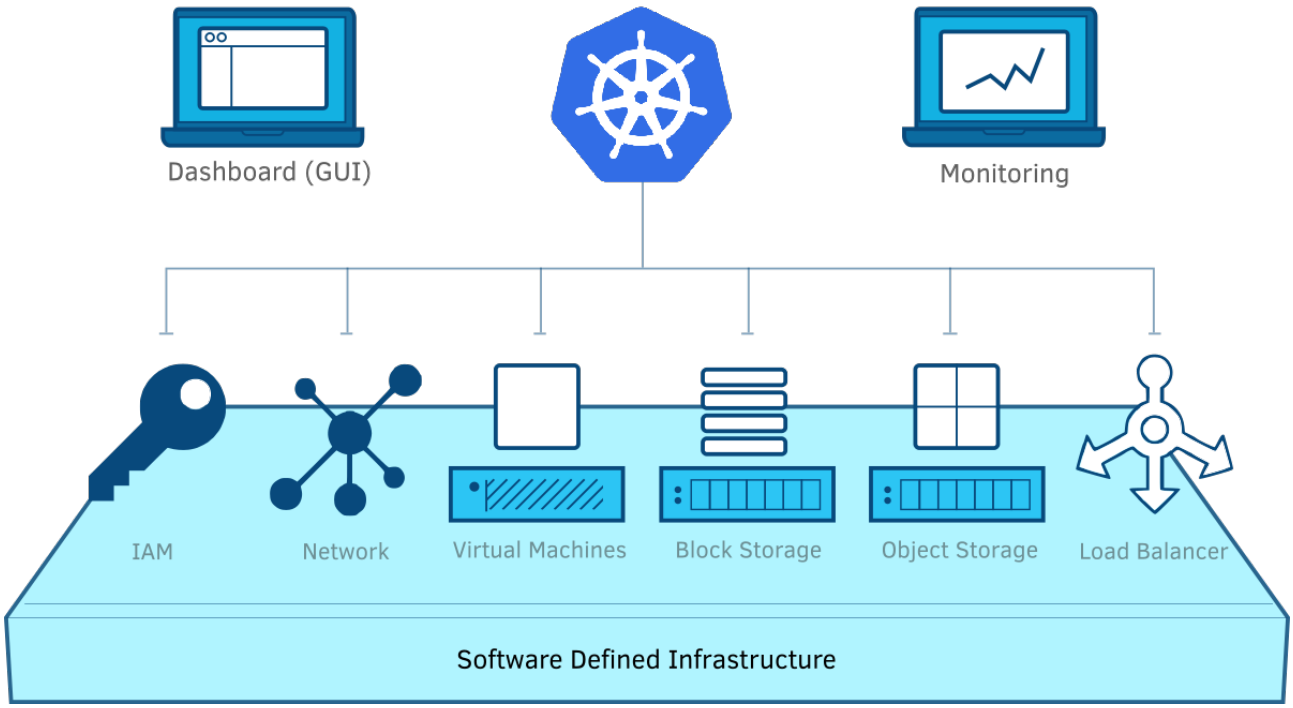
In the event of a major vulnerability, Catalyst Cloud will trigger a rolling upgrade on behalf of the user, so that the security update is applied as soon as possible. These security updates never

increase the minor or major version of Kubernetes. Users can opt-out from auto security updates at cluster creation time. For minor vulnerabilities, users can update the cluster (using upgrade API call) at a time convenient for their business.

To ensure applications are not impacted by updates and/or upgrades, users should follow the best practices outlined in the documentation: https://docs.catalystcloud.nz/kubernetes/rolling-upgrade.html#avoiding-application-downtime

3.3 Integration with Catalyst Cloud

Kubernetes is configured to integrate seamlessly with other Catalyst Cloud services. For example, when users request the creation of a persistent volume or an ingress, these requests are translated into API calls to the Catalyst Cloud Block Storage and Load Balancer services.



Identity and Access Management

Kubernetes is configured to authenticate users against the Catalyst Cloud IAM service (using a component called “k8s-keystone-auth” deployed to all master nodes of the cluster).

Network and security groups

Cloud software-defined networks, subnets, routers, and floating IPs are used to provide the network foundation required to deploy Kubernetes according to the cluster template selected and the configuration defined by the user.

Cloud security groups (akin to a software-defined firewall distributed across the entire network) are used to restrict the communication between Kubernetes components following hardening best practices.

Block storage

Pre-defined storage classes are present in Kubernetes enabling users to create persistent volumes using any of the block storage volume types available in the cloud. Persistent volume claims are managed by the OpenStack Cloud Controller.

Volumes are created, mounted, and deleted dynamically, according to the parameters specified in the volume claim. This integration is crucial, because pod placement is very dynamic and volumes must move together with pods as the Kubernetes scheduler places them in a new location.

Load balancer

Layer 4 and layer 7 load balancers are automatically created using the cloud load balancer service when users create a new service (with type=LoadBalancer) or a new ingress in Kubernetes. Layer 7 routing rules in the cloud load balancer are automatically updated when paths are introduced or modified by users on the ingress.

Logging and SIEM integration

Application logs, container logs, and Kubernetes logs can be easily forwarded to an object storage bucket, remote logging solution, or a SIEM, using Fluentd. For more information on how to configure logging targets, please refer to the documentation at: https://docs.catalystcloud.nz/kubernetes/logging.html