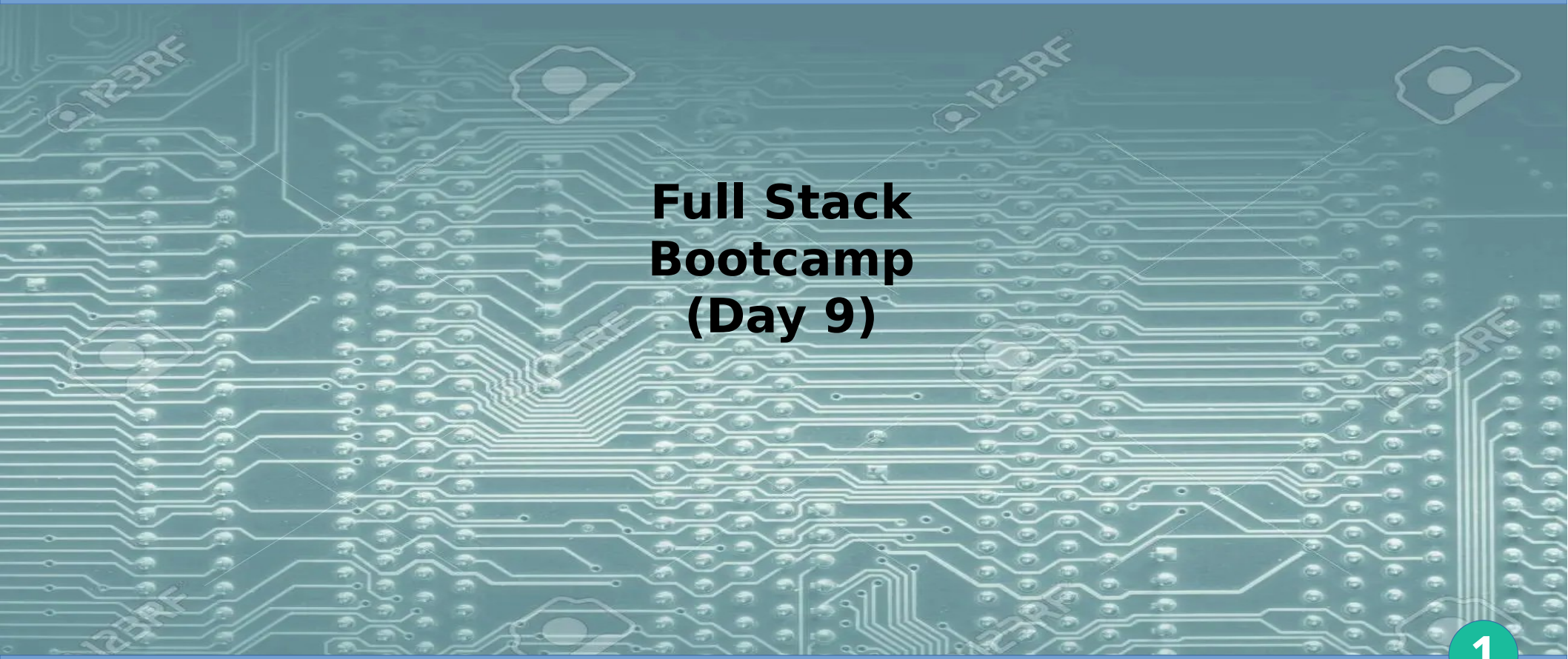
Kubernetes Fundamentals



Full Stack Bootcamp (Day 9)

Agenda

- Day 1
  - ES6+
- Day 2
  - React Native
- Day 3
  - Angular
- Day 4
  - Springboot
  - SpringData
- Day 5
  - JSON
  - NoSQL
- Day 6
  - Relational
- Day 7
  - Junit
  - Mockito
- Day 8
  - Docker
- Day 9
  - Kubernetes
- Day 10
  - Images and tips

Overview

- There’s no better overview than:
- https://www.youtube.com/watch?v=4ht22ReBjno

Basics

- Container orchestration is the automation of much of the operational effort required to run containerized workloads and services. This includes a wide range of things software teams need to manage a container’s lifecycle, including provisioning, deployment, scaling (up and down), networking, load balancing and more.
  - Networking
  - High availability
  - Ease of deployment & maintenance
  - Scalability
  - service discovery
  - Security & Compliance
  - Support (Community & Enterprise)
  - Administrative overhead

Networking

- The container orchestrators create an internal network
- Gateways
- Exposure of ports
- Load balancers

Microservices patterns

- Data
  - Database per service
- Deployment
  - Multiple services per host
- Crosscutting concerns
  - Microservice chassis
  - Externalized configuration
- Communication
  - API Gateway

Microservices patterns

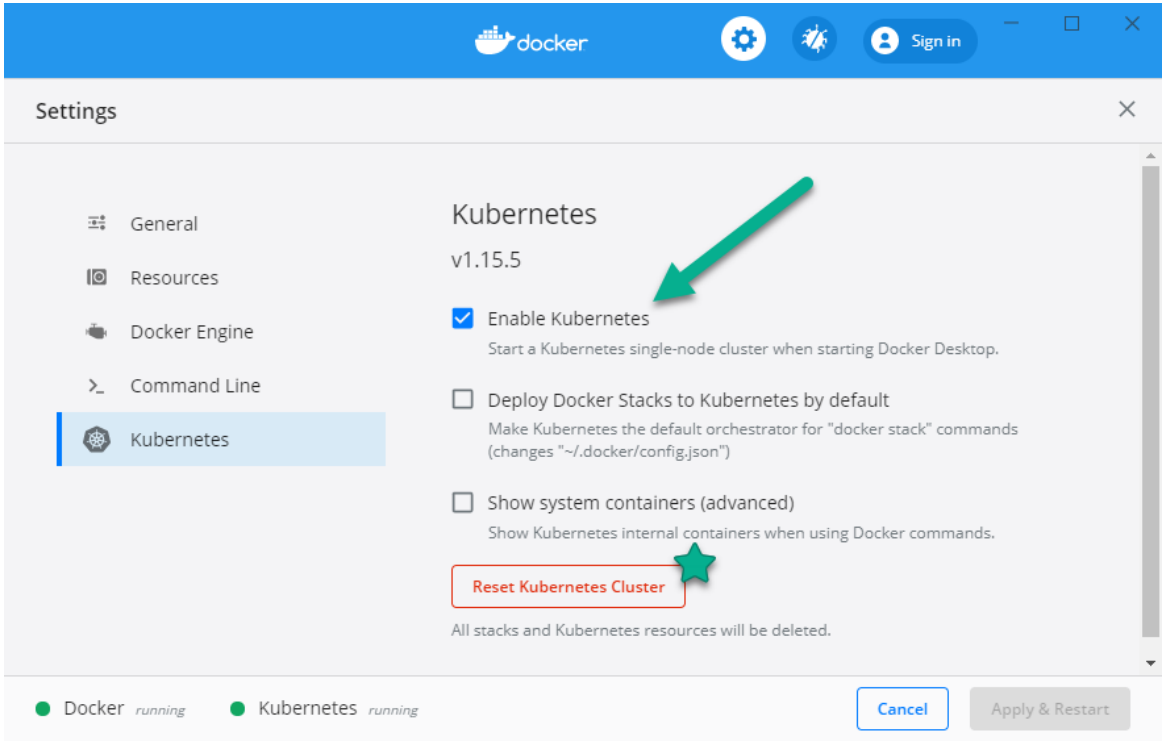
- Discovery
  - Client side discovery
  - Server side discovery
- Circuit breaker
- Security
  - Access token
- User Interface
  - Server side fragment composition
  - Client side UI composition

Microservices patterns

- Observability
  - Health check API
  - Distributed tracing
  - Exception tracking
  - Application logging
- Testing
  - Service component test
  - Service Integration Contract Test

Kubernetes

- Most popular orchestrator
- Provides seamlessly integration with Docker
- Basic services
  - Pods
  - Services
  - Bindings
  - Volumes
  - Nodes
  - Clustering



Setup

- Find your “Kubernetes” distribution, for personal use microk8s is recommended. https://microk8s.io/docs/install-alternatives
  - Download installer
  - Configure K8s
  - Test on command line

Post installation

- On windows microk8s starts automatically
- On linux start it from bash (command line)

```
charly@worker01:~$ sudo microk8s start
Started.
charly@worker01:~$ sudo microk8s status
microk8s is running
high-availability: no
  datastore master nodes: 127.0.0.1:19001
  datastore standby nodes: none
addons:
  enabled:
    ha-cluster           # (core) Configure high availability on the current node
  disabled:
    community            # (core) The community addons repository
    dashboard            # (core) The Kubernetes dashboard
    dns                  # (core) CoreDNS
    gpu                  # (core) Automatic enablement of Nvidia CUDA
    helm                 # (core) Helm 2 - the package manager for Kubernetes
    helm3                # (core) Helm 3 - Kubernetes package manager
    host-access          # (core) Allow Pods connecting to Host services smoothly
    hostpath-storage     # (core) Storage class; allocates storage from host directory
    ingress              # (core) Ingress controller for external access
    mayastor             # (core) OpenEBS MayaStor
    metallb              # (core) Loadbalancer for your Kubernetes cluster
    metrics-server       # (core) K8s Metrics Server for API access to service metrics
    prometheus           # (core) Prometheus operator for monitoring and logging
    rbac                 # (core) Role-Based Access Control for authorisation
    registry             # (core) Private image registry exposed on localhost:32000
    storage              # (core) Alias to hostpath-storage add-on, deprecated
```

Enabling dashboard

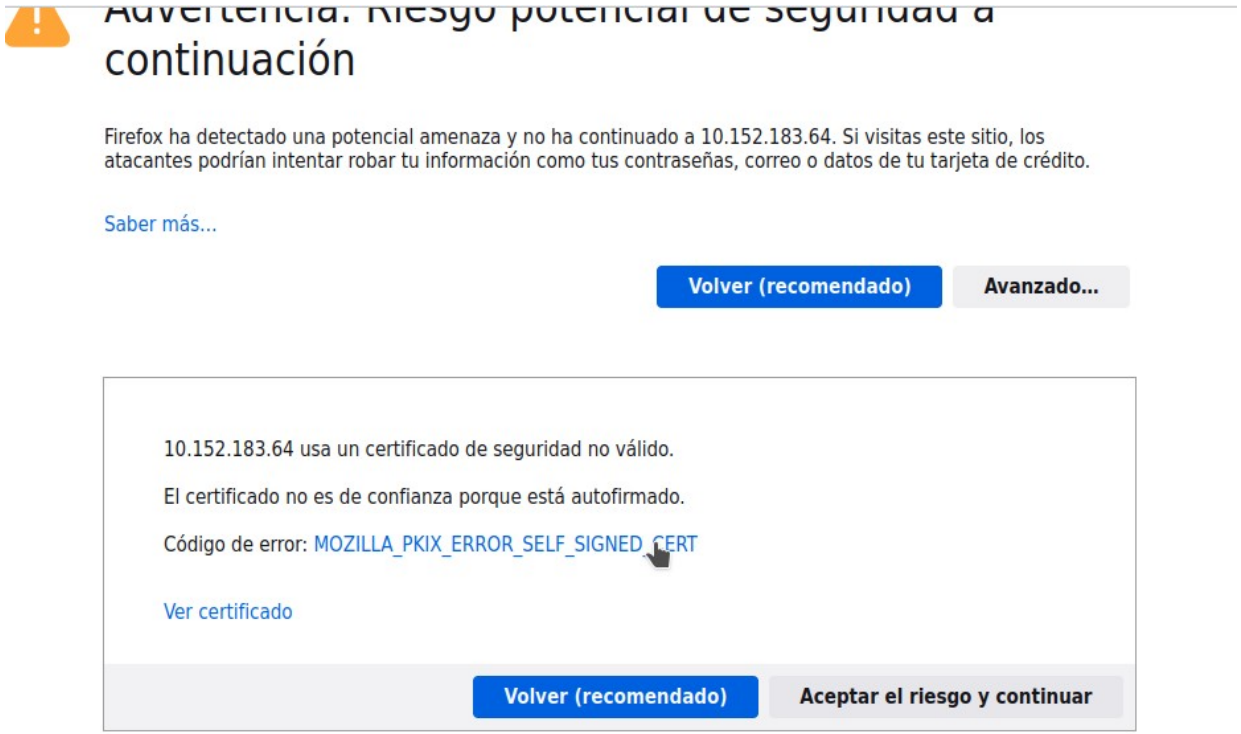
- On Linux super user powers are required

```
charly@worker01:~$ sudo microk8s enable dashboard
Infer repository core for addon dashboard
Enabling Kubernetes Dashboard
Infer repository core for addon metrics-server
Enabling Metrics-Server
serviceaccount/metrics-server created
clusterrole.rbac.authorization.k8s.io/system:aggregated-metrics-reader created
clusterrole.rbac.authorization.k8s.io/system:metrics-server created
rolebinding.rbac.authorization.k8s.io/metrics-server-auth-reader created
clusterrolebinding.rbac.authorization.k8s.io/metrics-server:system:auth-delegator created
clusterrolebinding.rbac.authorization.k8s.io/system:metrics-server created
service/metrics-server created
deployment.apps/metrics-server created
apiservice.apiregistration.k8s.io/v1beta1.metrics.k8s.io created
clusterrolebinding.rbac.authorization.k8s.io/microk8s-admin created
Metrics-Server is enabled
Applying manifest
serviceaccount/kubernetes-dashboard created
service/kubernetes-dashboard created
secret/kubernetes-dashboard-certs created
secret/kubernetes-dashboard-csrf created
secret/kubernetes-dashboard-key-holder created
configmap/kubernetes-dashboard-settings created
role.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrole.rbac.authorization.k8s.io/kubernetes-dashboard created
rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
deployment.apps/kubernetes-dashboard created
service/dashboard-metrics-scraper created
deployment.apps/dashboard-metrics-scraper created

If RBAC is not enabled access the dashboard using the default token retrieved with:

token=$(microk8s kubectl -n kube-system get secret | grep default-token | cut -d " " -f1)
microk8s kubectl -n kube-system describe secret $token

In an RBAC enabled setup (microk8s enable RBAC) you need to create a user with restricted
permissions as shown in:
https://github.com/kubernetes/dashboard/blob/master/docs/user/access-control/creating-sample-user.md
```

Access Token

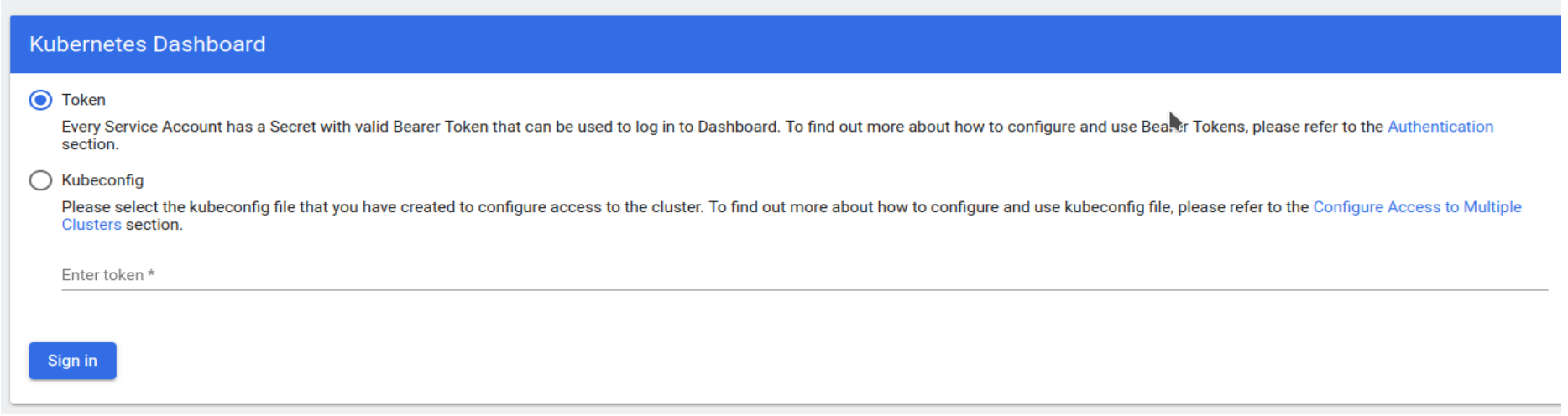
```
charly@worker01:~$ sudo microk8s dashboard-proxy
Checking if Dashboard is running.
Infer repository core for addon dashboard
Waiting for Dashboard to come up.
Create token for accessing the dashboard
secret/microk8s-dashboard-proxy-token created
Waiting for secret token (attempt 0)
Dashboard will be available at https://127.0.0.1:10443
Use the following token to login:
eyJhbGciOiJSUzI1NiIsImtpZCI6InZoNGxBSUJEcGJRMC1zNzhwbGU3RWg5eWsybGktN2RtMFZyU2ZMVXBYVWcifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VydmljZWFjY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6IjU5MGM5ZWExLWIwNGEtNDZiNi1iZGZhLTcxOWFhZDU1NWRmNiIsInN1YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDprdWJlLXN5c3RlbTpkZWZhdWx0In0.Ns1FJder2TtmLStA0HKSgDgDBwC1nM227vCZ7gQPSCPs17LGCHU5cCqQ0IvihNqobS110jUtx3AQojc35TDdKShn5B_ObD6aYPGIWKhSXglsQCYsL3pcjGR16RVOb0ewKmLHuwHgPQ2BzRBP6QrLC0PbU5kz2JK9-w-RX6ZZLVzE87IN2iPozq1J6Syo49PESwi_redW_SKRh9_FcI6UZPeE8v_cMjF8yKBGNm4llvvnV2MvMPifDLzVZm0QtG6Vo41RAIVXMEis1-zCtrlRpQTN00cWE_cv-YwPjhROe5E2ejNvb_P6HMqQOHfPYFCOdLqx6LKOxIA7xNMS6fpxIQ
```

Autosigned dashboard

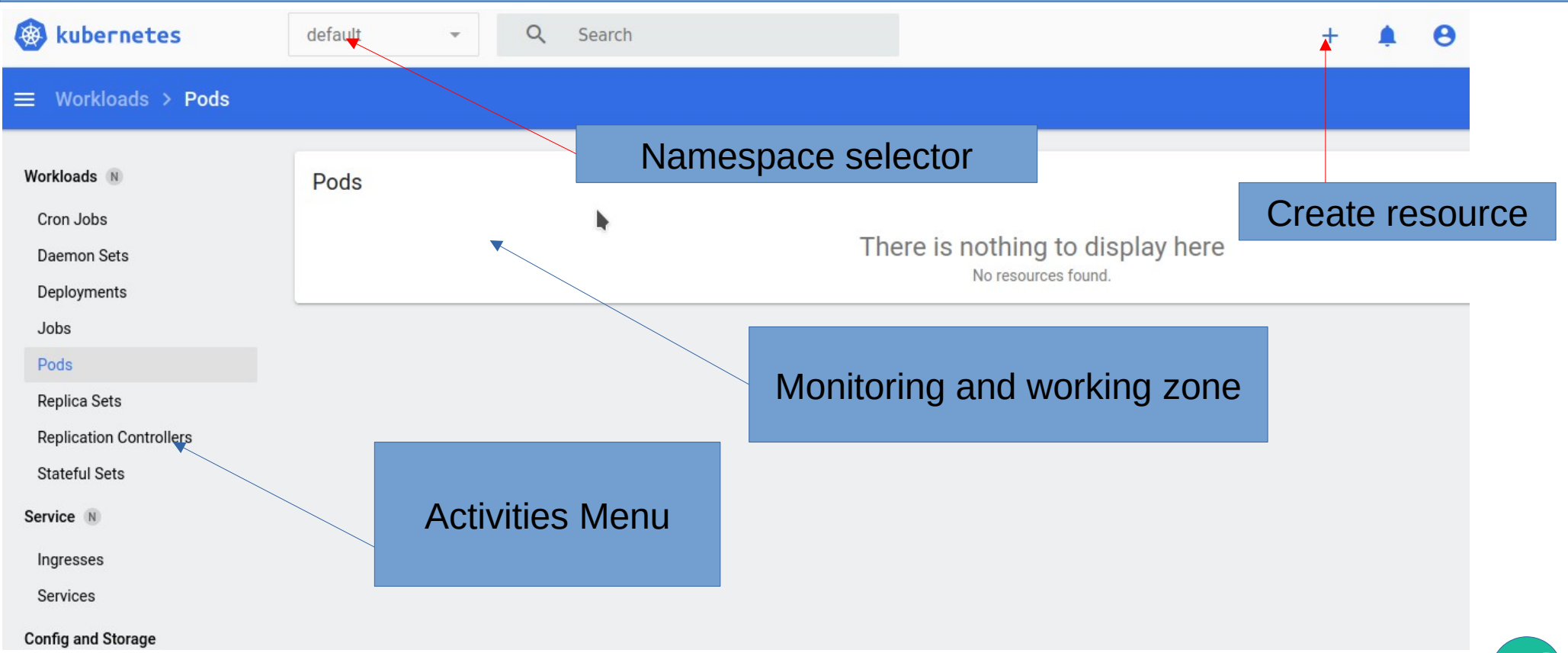


Dashboard access

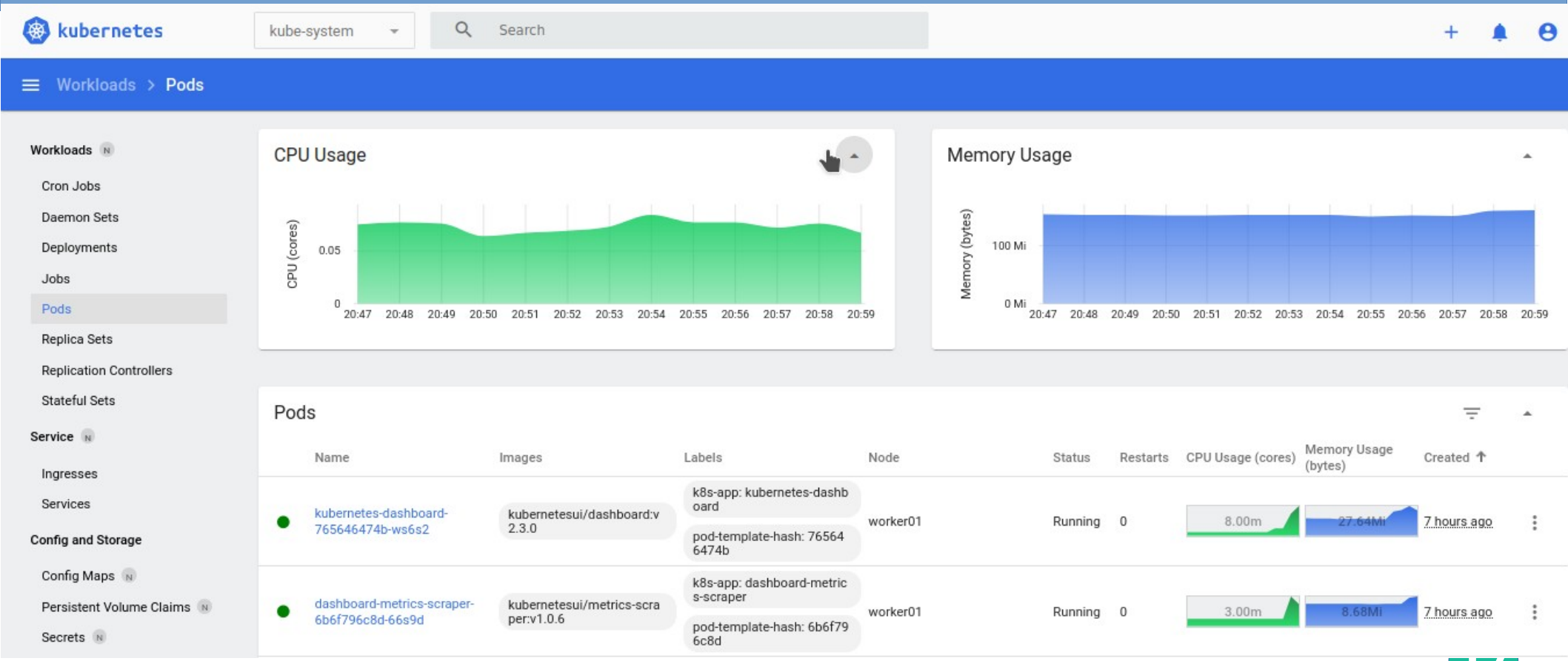
- **`sudo microk8s dashboard-proxy`**
- Copy token



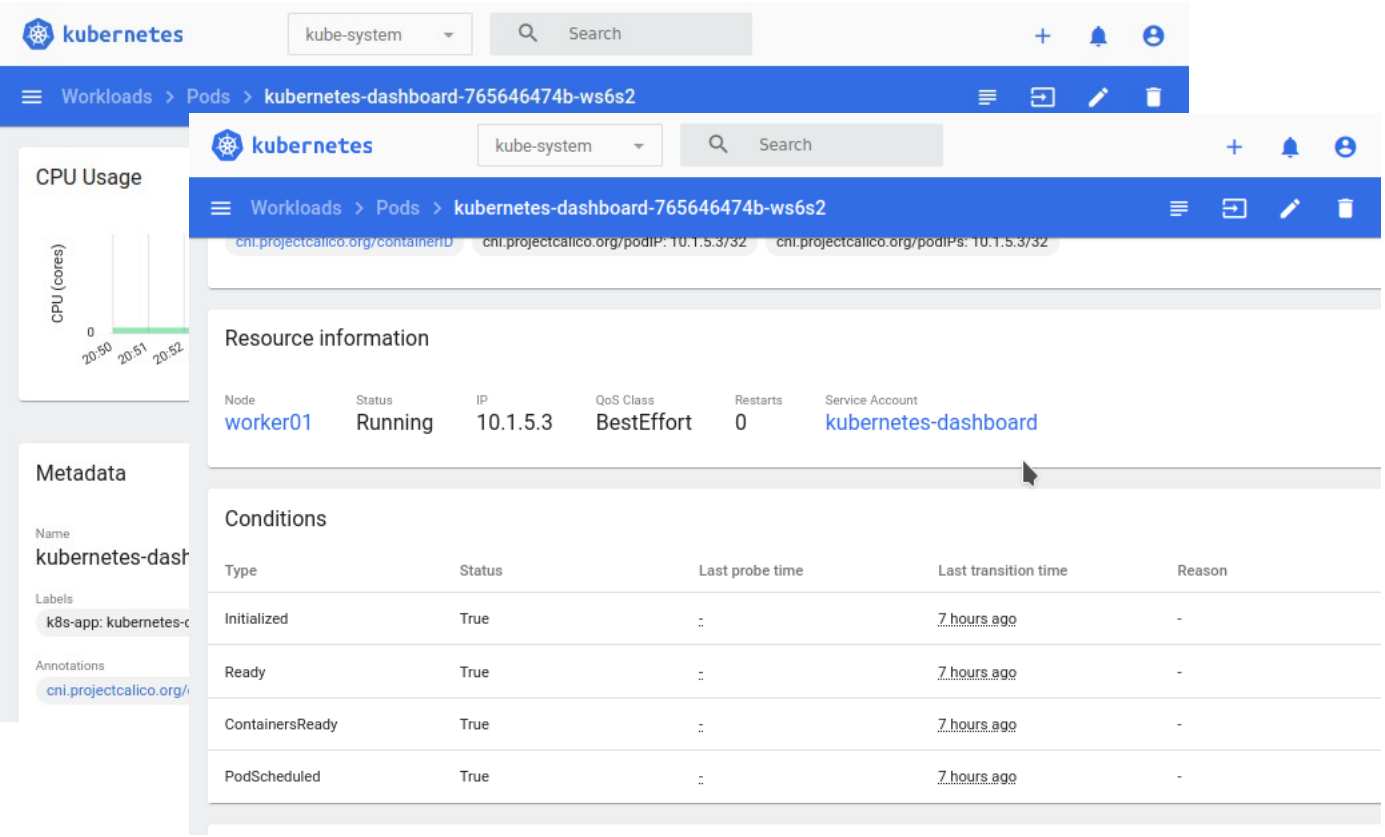
Dashboard


kubernetes
default
Search
Workloads > Pods
Workloads N
Cron Jobs
Daemon Sets
Deployments
Jobs
Pods
Replica Sets
Replication Controllers
Stateful Sets
Service N
Ingresses
Services
Config and Storage
Pods
There is nothing to display here
No resources found.
Namespace selector
Create resource
Monitoring and working zone
Activities Menu

Pods

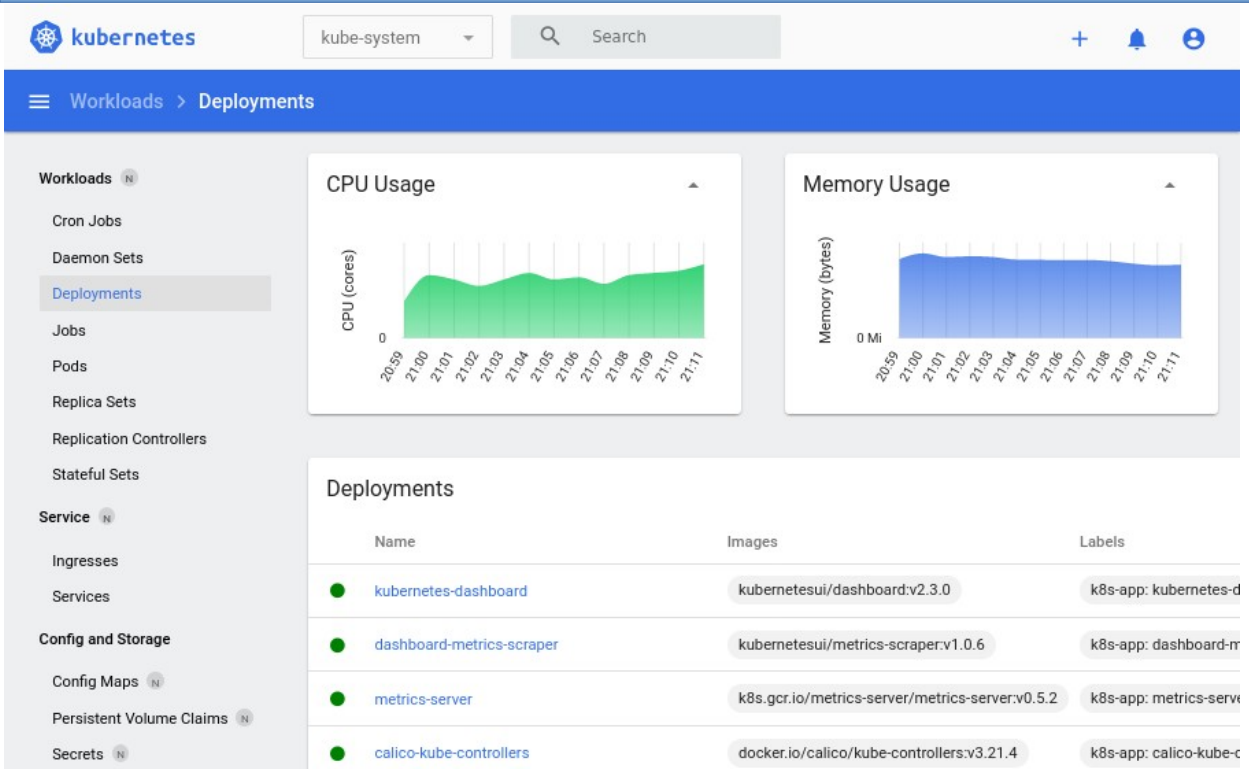

kubernetes
kube-system
Search
Workloads > Pods
Workloads
Cron Jobs
Daemon Sets
Deployments
Jobs
Pods
Replica Sets
Replication Controllers
Stateful Sets
Service
Ingresses
Services
Config and Storage
Config Maps
Persistent Volume Claims
Secrets
CPU Usage
CPU (cores)
0.05
0
20:47 20:48 20:49 20:50 20:51 20:52 20:53 20:54 20:55 20:56 20:57 20:58 20:59
Memory Usage
Memory (bytes)
100 Mi
0 Mi
20:47 20:48 20:49 20:50 20:51 20:52 20:53 20:54 20:55 20:56 20:57 20:58 20:59
Pods
Name
Images
Labels
Node
Status
Restarts
CPU Usage (cores)
Memory Usage (bytes)
Created
kubernetes-dashboard-765646474b-ws6s2
kubernetesui/dashboard:v2.3.0
k8s-app: kubernetes-dashboard
pod-template-hash: 765646474b
worker01
Running
0
8.00m
27.64Mi
7 hours ago
dashboard-metrics-scraper-6b6f796c8d-66s9d
kubernetesui/metrics-scraper:v1.0.6
k8s-app: dashboard-metrics-scraper
pod-template-hash: 6b6f796c8d
worker01
Running
0
3.00m
8.68Mi
7 hours ago

Pod details



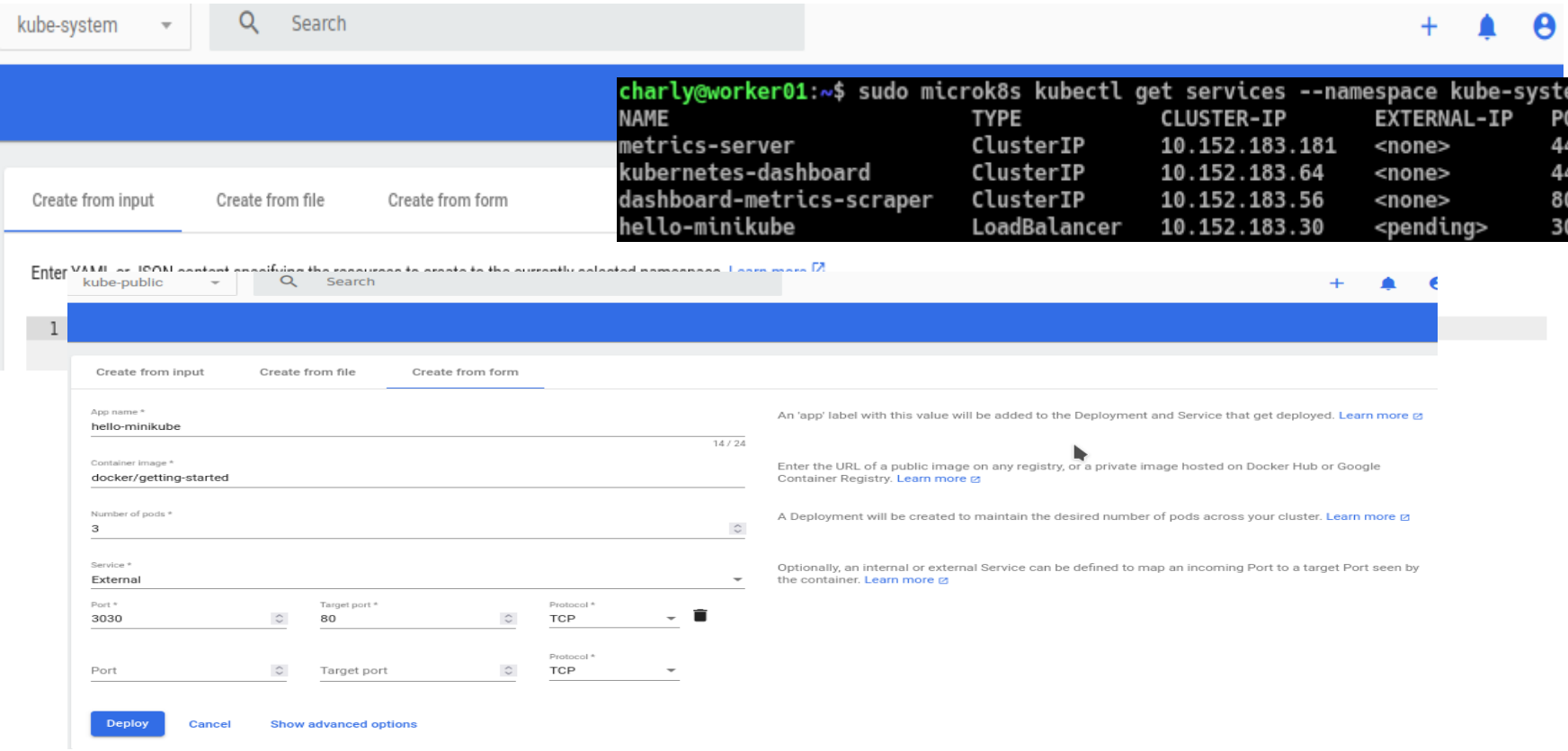
- Monitoring CPU and memory usage
- Metadata
- Resource information
- Conditions
- Controlled by
- Persistence volumes claims
- Events
- Containers
  - Image
  - Status
  - Arguments
  - Mounts
  - Security Context
  - Liveness probe

Deployments

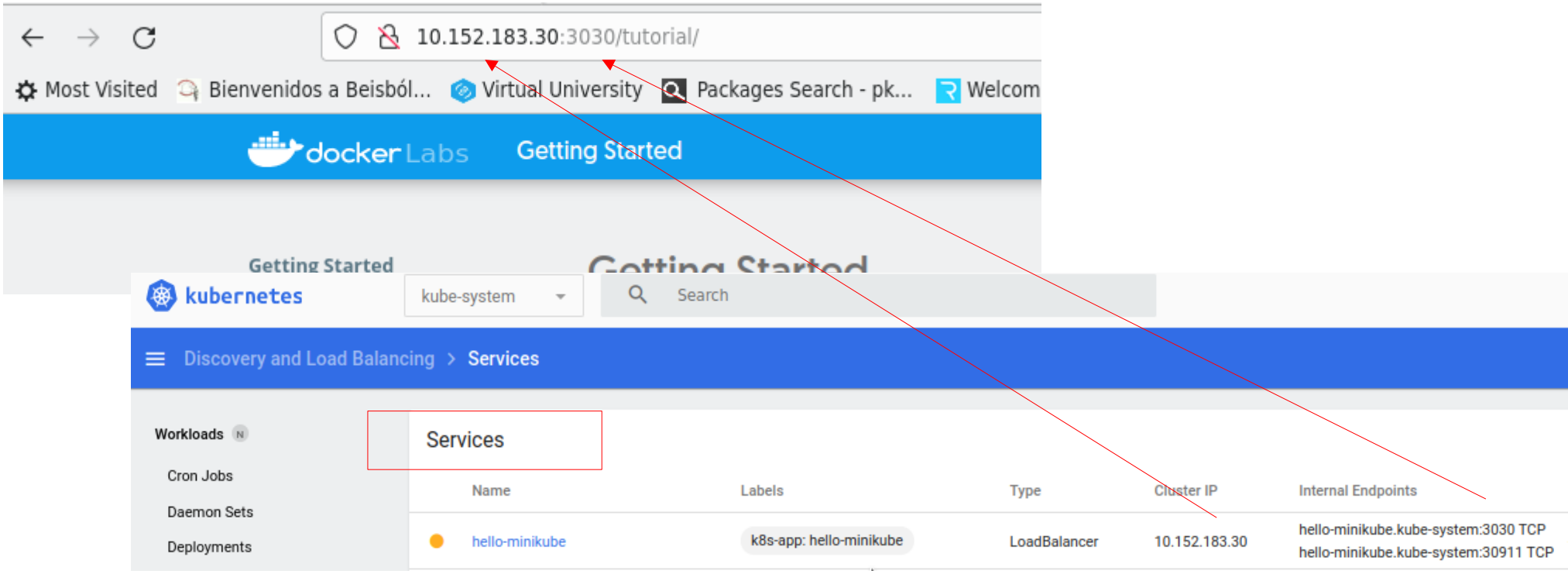


- pods. In short, a pod is the core building block for running applications in a Kubernetes cluster; a deployment is a management tool used to control the way pods behave.

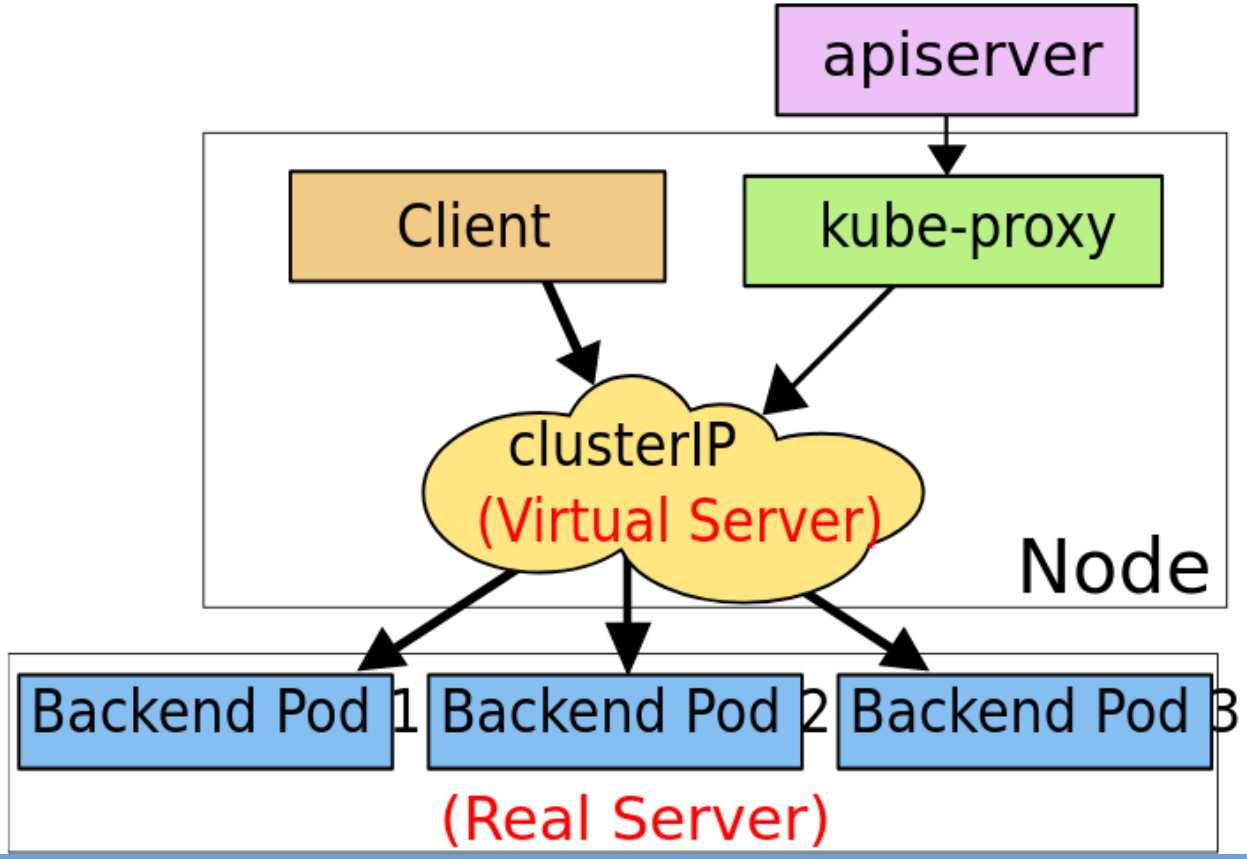
Hello miniKube



Hello Minikube



Configuration



- Service
- Pod
- DNS
- Deployments
- Namespaces
- Nodes

Clusters

- A Kubernetes cluster is a set of nodes that run containerized applications.
- Containerizing applications packages an app with its dependences and some necessary services. They are more lightweight and flexible than virtual machines.

Clusters

- A Kubernetes cluster contains six main components:
  - **API server:** Exposes a REST interface to all Kubernetes resources. Serves as the front end of the Kubernetes control plane.
  - **Scheduler:** Places containers according to resource requirements and metrics. Makes note of Pods with no assigned node, and selects nodes for them to run on.
  - **Controller manager**: Runs controller processes and reconciles the cluster's actual state with its desired specifications. Manages controllers such as node controllers, endpoints controllers and replication controllers.
  - **Kubelet:** Ensures that containers are running in a Pod by interacting with the Docker engine , the default program for creating and managing containers. Takes a set of provided **PodSpecs** and ensures that their corresponding containers are fully operational.
  - **Kube-proxy:** Manages network connectivity and maintains network rules across nodes. Implements the Kubernetes Service concept across every node in a given cluster.
  - **Etcd:** Stores all cluster data. Consistent and highly available Kubernetes backing store.

Clusters

- Automation occurs via the Pod Lifecycle Event Generator, or PLEG. These automatic tasks can include:
  - Starting and restarting containers
  - Adjusting the number of replicas for an application
  - Validating container images
  - Launching and managing containers
  - Implementing updates and rollbacks

Stateles Apps

- No storage assigned
- The app state is lost when **pod**, **namespace** or **kubernetes instance** is restarted
- Exposed to “the world”

Statefull Apps

- Creating a StatefulSet
- Begin by creating a StatefulSet using the example below. It is similar to the example presented in the StatefulSets concept. It creates a headless Service, nginx, to publish the IP addresses of Pods in the StatefulSet, web.

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    app: nginx
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  serviceName: "nginx"
  replicas: 2
  selector:
    matchLabels:
      app: nginx
template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: k8s.gcr.io/nginx-slim:0.8
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 1Gi
```

State-full Apps

- Using the yaml file create the state-full app

  ```
  kubectl apply -f web.yaml
  ```

- Review service status

  ```
  kubectl get service nginx
  ```

- StatefulSets concept, the Pods in a StatefulSet have a sticky, unique identity. This identity is based on a unique ordinal index that is assigned to each Pod by the StatefulSet controller.

  ```
  kubectl get pods -l app=nginx
  ```

- Each Pod has a stable hostname based on its ordinal index. Use **`kubectl`** exec to execute the **`hostname`** command in each Pod:

  ```
  for i in 0 1; do kubectl exec "web-$i" -- sh -c 'hostname'; done
  ```

- Get the PersistentVolumeClaims for web-0 and web-1:

  ```
  kubectl get pvc -l app=nginx
  ```

Statefull Apps

- The **StatefulSet** controller created two **PersistentVolumeClaims** that are bound to two **PersistentVolumes**.
- As the cluster used in this tutorial is configured to dynamically provision **PersistentVolumes**, the **PersistentVolumes** were created and bound automatically.
- The **NGINX** webserver, by default, serves an index file from /usr/share/nginx/html/index.html. The **volumeMounts** field in the **StatefulSet's** spec ensures that the `/usr/share/nginx/html` directory is backed by a **PersistentVolume**.
- Write the Pods' hostnames to their index.html files and verify that the **NGINX** webservers serve the hostnames:

```
for i in 0 1; do kubectl exec "web-$i" -- sh -c 'echo "$(hostname)" > /usr/share/nginx/html/index.html'; done
```

```
for i in 0 1; do kubectl exec -i -t "web-$i" -- curl http://localhost/; done
```

Statefull Apps

- Scaling a **StatefulSet** refers to increasing or decreasing the number of replicas. This is accomplished by updating the replicas field. You can use either **`kubectl`** scale or **`kubectl`** patch to scale a **StatefulSet**.
- Scaling Up
  - In one terminal window, watch the Pods in the **StatefulSet**:

    ```
    kubectl get pods -w -l app=nginx
    ```

  - In another terminal window, use **`kubectl`** scale to scale the number of replicas to 5:
    - **`kubectl scale sts web --replicas=5`**

Services

Kubernetes networking addresses four concerns:

- Containers within a Pod use networking to communicate via loopback.
- Cluster networking provides communication between different Pods.
- The Service resource lets you expose an application running in Pods to be reachable from outside your cluster.
- You can also use Services to publish services only for consumption inside your cluster.

Services

- A serviceis an abstract way to expose an application running on a set of Pods as a network service.
- With Kubernetes you don't need to modify your application to use an unfamiliar service discovery mechanism. Kubernetes gives Pods their own IP addresses and a single DNS name for a set of Pods, and can load-balance across them
- If you're able to use Kubernetes APIs for service discovery in your application, you can query the API server for Endpoints, that get updated whenever the set of Pods in a Service changes.
- For non-native applications, Kubernetes offers ways to place a network port or load balancer in between your application and the backend Pods.

Storage

- Volumes
  - On-disk files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers. One problem is the loss of files when a container crashes. The kubelet restarts the container but with a clean state. A second problem occurs when sharing files between containers running together in a Pod. The Kubernetes volume abstraction solves both of these problems
- Type of volumes
  - Persistent
  - Projected
  - Ephemeral

Volumes

- A PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes. It is a resource in the cluster just like a node is a cluster resource. PVs are volume plugins like Volumes, but have a lifecycle independent of any individual Pod that uses the PV. This API object captures the details of the implementation of the storage, be that NFS, iSCSI, or a cloud-provider-specific storage system.
- A PersistentVolumeClaim (PVC) is a request for storage by a user. It is similar to a Pod. Pods consume node resources and PVCs consume PV resources. Pods can request specific levels of resources (CPU and Memory). Claims can request specific size and access modes (e.g., they can be mounted ReadWriteOnce, ReadOnlyMany or ReadWriteMany, see AccessModes).

Volumes

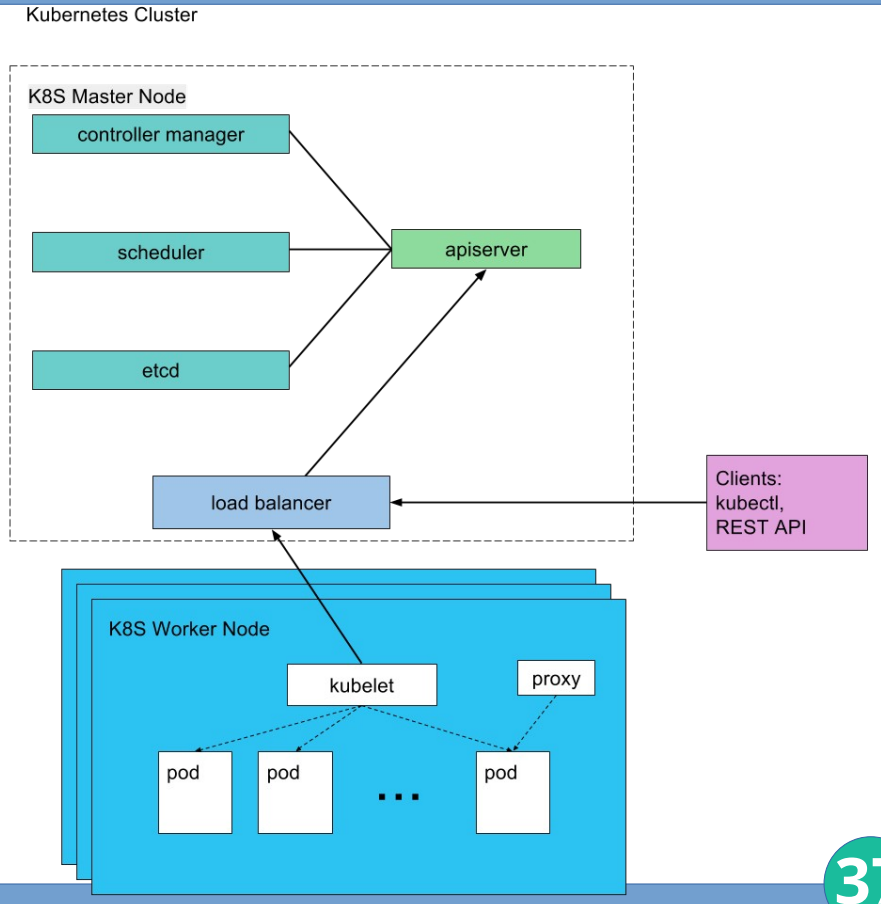
- A projected volume maps several existing volume sources into the same directory.
- Currently, the following types of volume sources can be projected:
  - secret
  - downwardAPI
  - configMap
  - serviceAccountToken
- All sources are required to be in the same namespace as the Pod.

Volumes

- Some application need additional storage but don't care whether that data is stored persistently across restarts. For example, caching services are often limited by memory size and can move infrequently used data into storage that is slower than memory with little impact on overall performance.
- Other applications expect some read-only input data to be present in files, like configuration data or secret keys.
- Ephemeral volumes are designed for these use cases.

High availability

- Kubernetes High Availability ensures that Kubernetes and its supporting components have no single point of failure. A single master cluster is vulnerable to failure, but a multi-master cluster uses many master nodes, each having access to the same worker nodes.
- Create Highly Available Kubernetes Clusters
  - Make the master node services reliable.
  - Set up a redundant storage layer for etcd.
  - Use a highly available load balancer for the Kubernetes API services.
  - Setup multiple master nodes and configure a master election strategy.



Day 9 summary

- Kubernetes overview
- Microservices patterns
- Setup Configuration
- Dashboard
- Image usage
- Clusters
- Stateless and Statefull apps
- Services
- Storage
- High availability