CREDERA

Deploying Code with Kubernetes

February 22, 2020

OVERVIEW

- Deployment Strategies:
  - Bare OS
  - VMs
  - Docker
    - Image & Container
  - Container Orchestration
- Kubernetes Core Concepts
  - Pod
  - Manifest
  - Namespace
  - Deployment
  - Services
  - Ingress
- Additional Concepts


DEPLOYING YOUR CODE

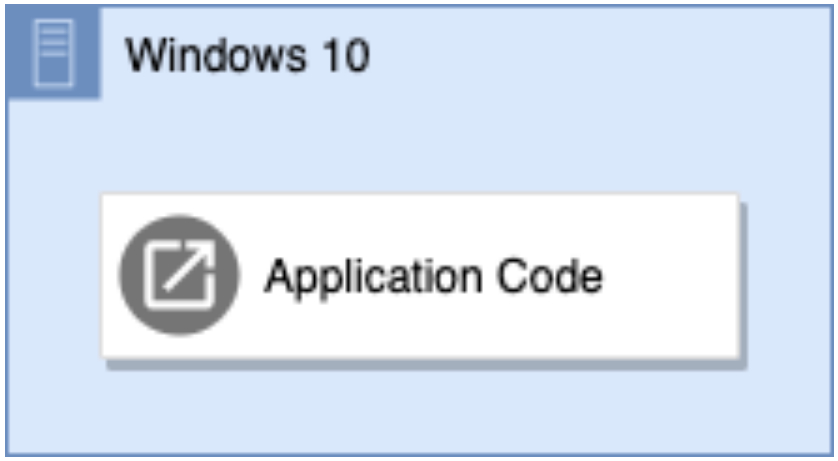
BARE OS – AS EASY AS IT GETS

HOW DOES IT WORK?

You do this every time that you compile and run your code locally

Application code is compiled into OS specific binaries (.exe's for windows) and those binaries are placed on the target machine

Those binaries are executed, and the application runs directly on the host OS



PROS

Extremely simple and approachable

Good for IoT

Cons

Application is specific to host environment

Not scalable

Potentially unused resources

Expensive

VMS – THE RIGHT DIRECTION

HOW DOES IT WORK?

Very common approach for both on premise and cloud environments

Application code is compiled into OS specific binaries (.exe's for windows) and those binaries are placed on a Virtual Machine running on a hypervisor

Those binaries are executed, and the application runs on the guest operating system

PROS

Still very simple

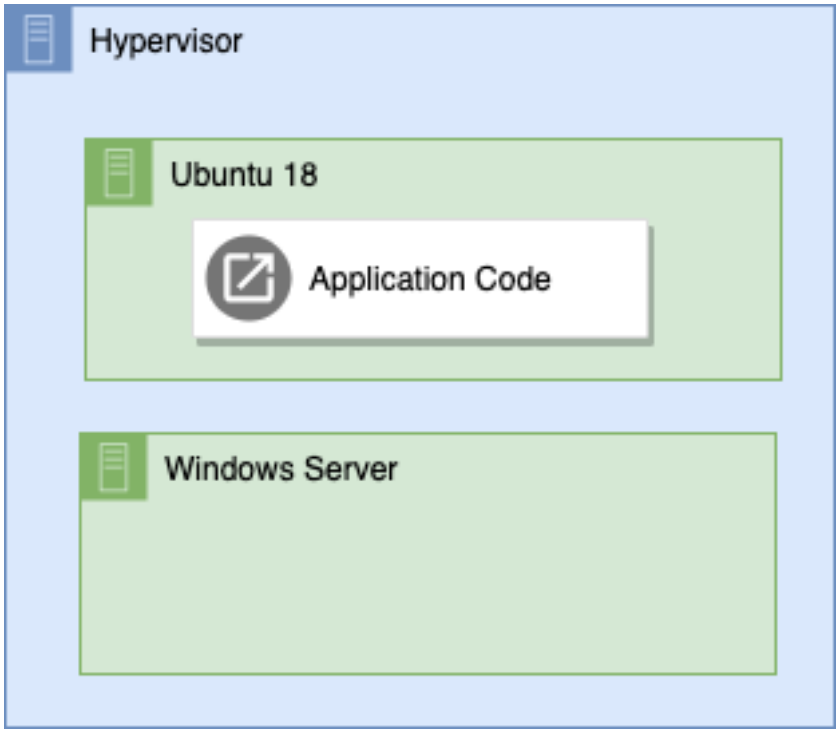
Sharing space with other VMs

Cons

Application is specific to guest OS

Not scalable

Dependency management



DOCKER – IMAGE & CONTAINER

IMAGE

A Docker Image is an immutable definition to build a Docker Container

An image is assembled from multiple layers, starting with a root image, and then applying modifications to that image

As an example, the following Dockerfile describes how to build a container for a node web application

```
FROM node:10

# Create app directory
WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install

# Bundle app source
COPY . .

EXPOSE 8081
CMD [ "npm", "start" ]
```

CONTAINER

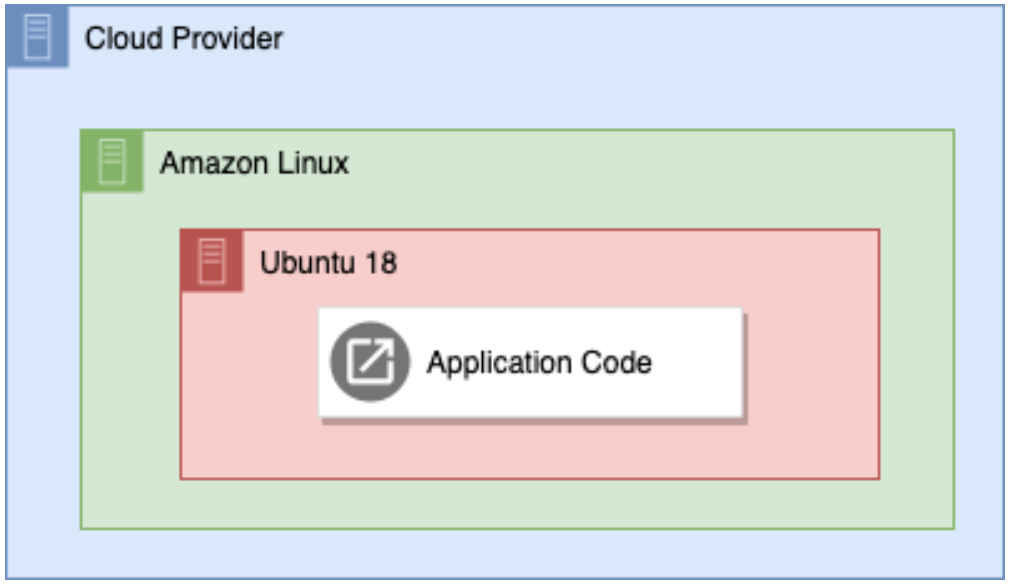
A Docker Container is the instantiation (execution) of a Docker Image, resulting in a fully formed environment on which to run an application

DOCKER – COOKING WITH GAS

HOW DOES IT WORK?

Application code is bundled along with supporting OS features into an Image, which is portable across anything that can run docker

Image is run as a Container, executing application code independently of the host OS



PROS

Extremely portable

Space efficient

Scalable

Cons

No coordination

Requires access to individual VM to deploy

CONTAINER ORCHESTRATION

HOW DOES IT WORK?

Application code is bundled along with all supporting OS features into an Image, which is portable across anything with docker installed

Manifest file is created, outlining how the application should be run, how many applications should run, etc.

Manifest file is parsed, and Containers are run independently of host OS

PROS

Extremely portable

Space efficient

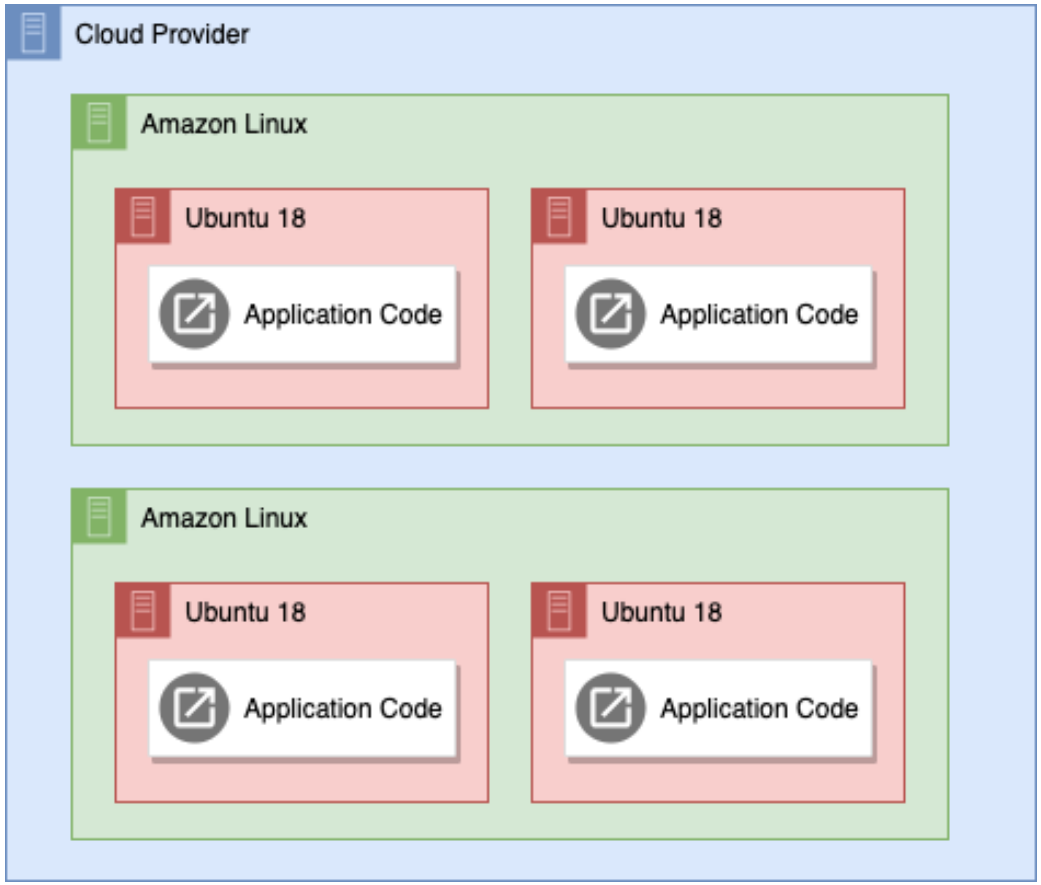
Auto Scalable

Hands off management

Declarative

Cons

Infrastructure overhead



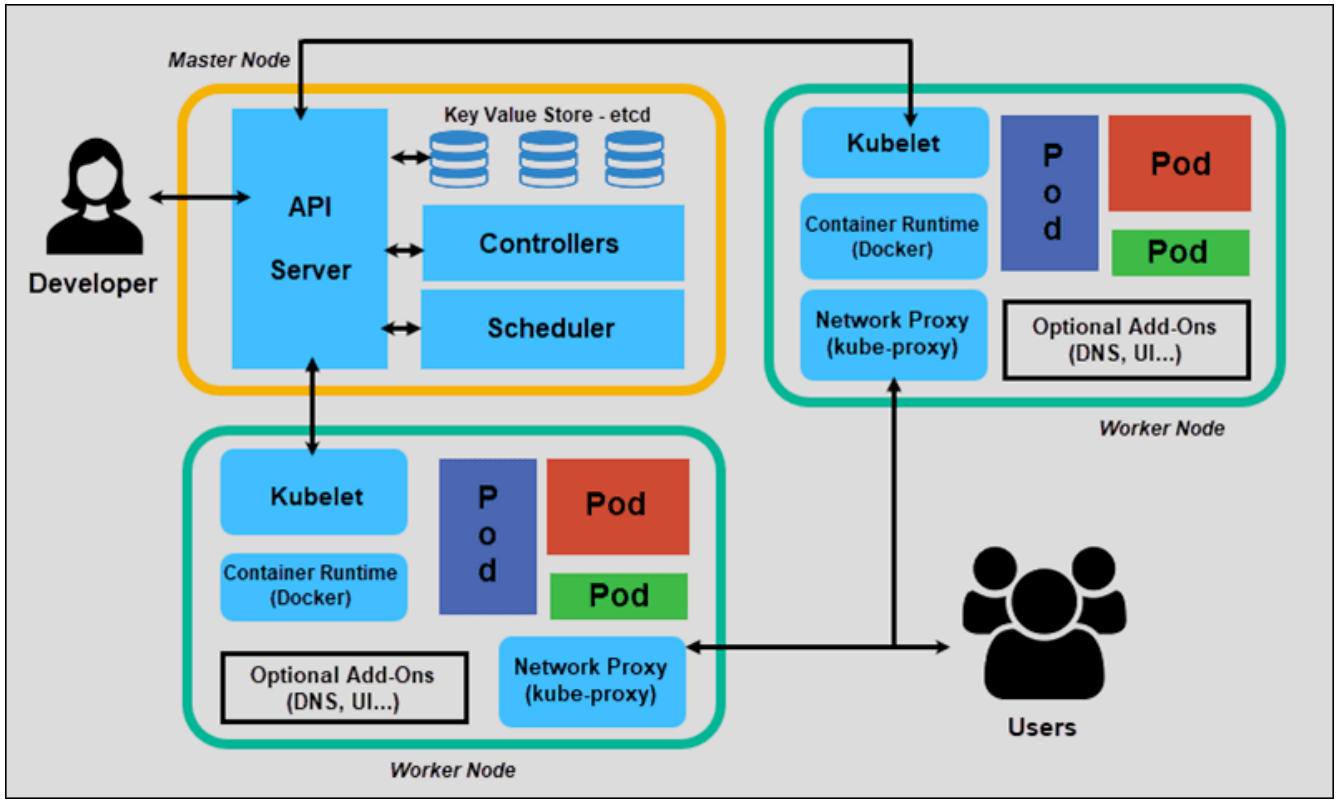
KUBERNETES CORE CONCEPTS

KUBERNETES OVERVIEW

DESCRIPTION

“Containers are a good way to bundle and run your applications. In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime. For example, if a container goes down, another container needs to start. Wouldn’t it be easier if this behavior was handled by a system?

That’s how Kubernetes comes to the rescue! Kubernetes provides you with a framework to run distributed systems resiliently. It takes care of scaling and failover for your application, provides deployment patterns, and more. For example, Kubernetes can easily manage a canary deployment for your system.”



MANIFEST

DESCRIPTION

The Kubernetes manifest is a fully formed definition of an application to be managed by Kubernetes

Manifests are defined in a single file in yaml format, named manifest.yml and are stored in source control. Manifest files may contain the following sections:

- **Namespace**
- **Deployment**
- **Service**
- **Ingress**
- **Service Monitor**
- **Horizontal Pod Autoscaler**
- **Volume Claims**
- **Volume Definitions**
- **Secrets**
- **Config Maps**

Each section of the manifest specifies its own version

NAMESPACE

DESCRIPTION

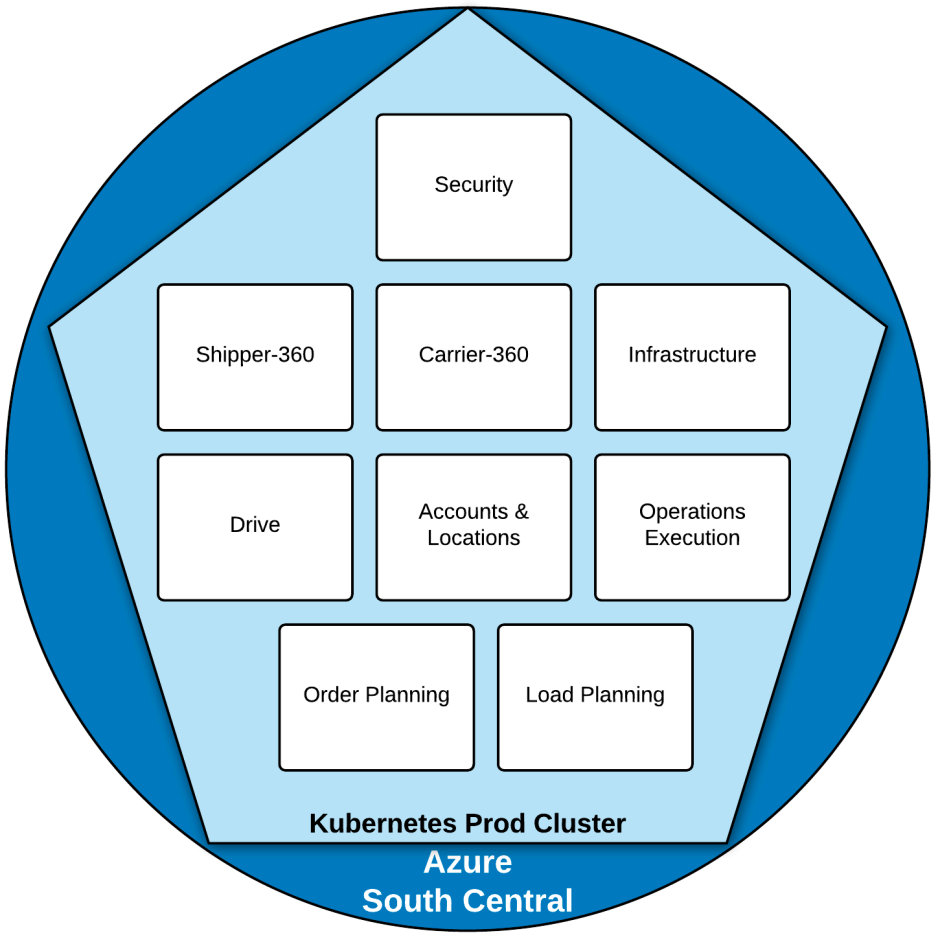
In Kubernetes, a Namespace is a logical grouping of applications forming a virtual cluster inside of a physical cluster

Namespaces can be used to break up the Kubernetes cluster by product, as seen in the diagram at right with each white box representing a different product's namespace

The Kubernetes cluster also initially defines three namespaces:

- default – default namespace
- kube-system – Kubernetes system objects
- kube-public – public readable namespace

```
kind: Namespace
apiVersion: v1
metadata:
  name: node-app
  labels:
    name: node-app
```

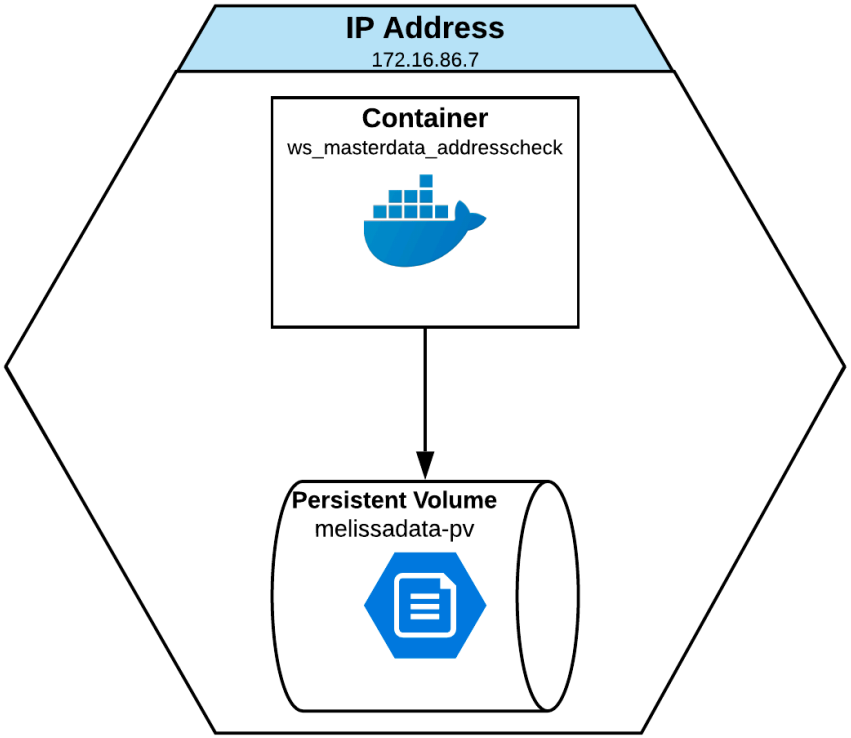


POD

DESCRIPTION

A pod is a logical combination of one or more containers, networking and storage designed to model a logical host, such as an entire VM that would previously run Windows, Tomcat and one or more web applications

Below is an example diagram of a Pod:



DEPLOYMENT

DESCRIPTION

A Kubernetes deployment defines the desired state of pods and replica sets (a set of identical pod)

Application teams will generally customize the following for their deployment:

- Number of replicas
- Container configuration
  - Resource requests and limits
  - Environment variables
  - Liveness and readiness probe definitions
- Storage volume mount points and access tokens

REPLICAS

The deployment specification configures the initial number of replicas for a given application
The number of running instances can grow beyond the configured number by using manual scaling from the command line or by using a horizontal pod autoscaler

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
   name: node-app
spec:
   replicas: 2
   template:
      metadata:
         labels:
            app: node-app
      spec:
         containers:
            - name: node-app
         image: node-app
            ports:
               - containerPort: 8081
```

SERVICE

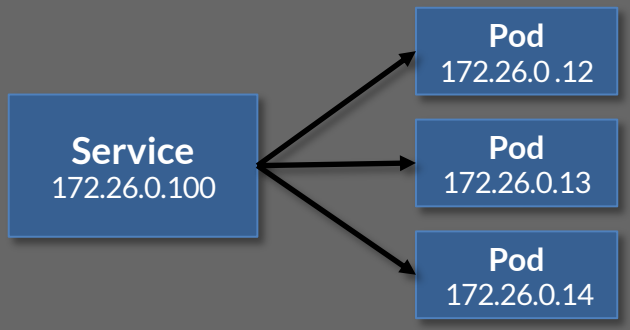
DESCRIPTION

A service is a proxy that sits in front of one or more Pods, which have their own way of being accessed, providing a stable way from which to access the Pods being proxied; services register Pods by matching label selectors

Service definitions can also specify protocol (TCP, UDP, SCTP) and target port for the destination pod

For example, the following service definition, **node-app-service**, is looking for any pods with the label of **node-app**. Once it finds those pods it can load balance across them

```yaml
apiVersion: v1
kind: Service
metadata:
  name: node-app-service
    labels:
      run: node-app-service
spec:
  ports:
  - port: 8081
    protocol: TCP
  selector:
    run: node-app
```



INGRESS

DESCRIPTION

A Kubernetes ingress defines a rule or set of rules by which an external application can access services in the cluster

In this example, we create the ingress **node-app-ingress** which exposes our node-app service to be accessed from outside of the cluster

Visit http://localhost to check it out.

```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: node-app-ingress
spec:
  backend:
    serviceName: node-app
    servicePort: 8081
```


Q&A