

<Kubernetes>

<Apex/Group/Operational> Layer

Document Version / Details: Ver. 2.1/ 02-Jan-2023

WORD DOCUMENT

Kubernetes

Heal -> kubernetes control and fix the damage

AutoHealing -> Start a new container app

API Server ->

If the container is going down before the first container goes down second container is already rollout

Control Plane/Controller Data Plane / Worker

API Server Kubelet

Etcd Proxy

Scheduler Container Runtime

Kubernetes production used in distribution

EKS

Openshift

Tanzu

Ranctier

AKS

GKE

DKE

Staging

Pre-prod 🡨---- K8s

On local

Kops -> most widely used tools for installing kubernetes

kubeadm

Installation

Upgrade

Deletions

Benefits of Kubernetes

1. Cluster
2. Scaling
3. Healing
4. Enterprise

In kubernestes lowest level of deployment is a pod

Docker run

-d -t ->

-p -> port mapping

-v -> volume

-network ->

Pod.yaml ->

Install ansible on the top of it

1. **Kubernetes Installation Using KOPS on EC2**

**kOps, which is a tool for automating the provisioning of Kubernetes clusters on AWS and GCE.**

1. **Create an EC2 instance or use your personal laptop.**

Dependencies required

1. Python3
2. AWS CLI
3. kubectl
4. **Install dependencies**

curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a /etc/apt/sources.list.d/kubernetes.list

sudo apt-get update

sudo apt-get install -y python3-pip apt-transport-https kubectl

pip3 install awscli --upgrade

export PATH="$PATH:/home/ubuntu/.local/bin/"

1. **Install KOPS (our hero for today)**

curl -LO https://github.com/kubernetes/kops/releases/download/$(curl -s https://api.github.com/repos/kubernetes/kops/releases/latest | grep tag\_name | cut -d '"' -f 4)/kops-linux-amd64

chmod +x kops-linux-amd64 //mark it as executable

sudo mv kops-linux-amd64 /usr/local/bin/kops

1. **Provide the below permissions to your IAM user. If you are using the admin user, the below permissions are available by default**
2. AmazonEC2FullAccess
3. AmazonS3Fu/llAccess
4. IAMFullAccess
5. AmazonVPCFullAccess
6. **AWS CLI configuration on your EC2 Instance or Laptop.**

Run aws configure

1. **Kubernetes Cluster Installation**

Please follow the steps carefully and read each command before executing.

1. **Create S3 bucket for storing the KOPS objects.**

aws s3api create-bucket --bucket kops-abhi-storage --region us-east-1

1. **Create the cluster**
   1. kops create cluster --name=demok8scluster.k8s.local --state=s3://kops-abhi-storage --zones=us-east-1a --node-count=1 --node-size=t2.micro --master-size=t2.micro --master-volume-size=8 --node-volume-size=8**Kubernetes Cluster Installation**

Please follow the steps carefully and read each command before executing.

If using cloud ->

kops create cluster --name=demok8scluster.amazon.com --state=s3://kops-abhi-storage --zones=us-east-1a --node-count=1 --node-size=t2.micro --master-size=t2.micro --master-volume-size=8 --node-volume-size=8**Kubernetes Cluster Installation**

**Important: Edit the configuration as there are multiple resources created which won't fall into the free tier.**

**kops edit cluster myfirstcluster.k8s.local**

**Step 12: Build the cluster**

**kops update cluster demok8scluster.k8s.local --yes --state=s3://kops-abhi-storage**

**This will take a few minutes to create............**

**After a few mins, run the below command to verify the cluster installation.**

**kops validate cluster demok8scluster.k8s.local**

docker run -d

-p -> port mapping

-network -> mount volume

All of the above things you need in docker

In kubernetes we need

Pod.yml file ->

Pod -> single container

* + multiple container

Building a pod with one single container

apiversion:v1

spec:

- containers:

Yaml -> Enterprise

Declarative

Yaml -> pod deployment -> example

Pod -> | (or) group container

Side -car

Init container

Application deployed in the container

Config ->

File

Building a pod with multiple container

* + A and B container in a single pod.
  + Shared networking
  + Shared storage

Single pod talk to localhost -> localhost:3000

Let’s say both of them want to share a file -> so in a single pod we are putting multiple container

But generally it is not practiced that way -> we put sidecar containers or init container

You can access the application inside the containers using this pod -> cluster IP.

IP address are generated for the pods ->

So Ip address are not generated for the container -> but they are generated for the pod.

If you go to git -> pod.yml -> you will understand about the container ->

So this container is running on the application is running inside it on port 80. It has a volume mount then you know what is networking ->

So basicaly you are accessing the container using the cluster ip address that kubernetes gave for pod so who is giving the cluster ip address ->

Kubeproxy -> is generating this cluster IP address ->

Kubectl -> command line for ctl

Kubectl get nodes -> to know how many nodes are present inside the kubernetes cluster

Kubectl

Minikube -> Kops ->

Download minikube, K3s, kind, microk8s

Kind is better than minikube

Basically kubernetes in docker -> kubernetes nodes or kubernetes entire setup is done same as docker container ->

Container Pod Deploy

Docker run -it Instead of writting all these things in

Docker run -d cmd we can write a file pod.yaml.

Docker run -p pod.yaml file contain

Docker run -v running specification

Docker run -network So inside a single pod there are multiple

Container.

Docker command Why do we create multiple containers in

a pod.

Pod

Let’s say you have an application that is dependent on other application without which it cannot run. You know you have a container here.

[Day-16 | AKS vs Self Managed Kubernetes Clusters | Pros and Cons | Azure Interview Questions (youtube.com)](https://www.youtube.com/watch?v=o_7yvVqLZXQ)

Creating AKS cluster on Azure

Azure Kubernetes Services (AKS Deep Dive)

AKS Deep Dive

Understanding AKS vs Self managed Kubernetes clusters

3 ways to create a kubernetes cluster

1. On-premise

Data Center

Openstack

A whiteboard with text and words

Description automatically generated

There is control plane and data plane in kubernetes architecture

Control plane are the master nodes -> 3 nodes

Data plane are the Worker nodes -> 2 nodes

1. Virtual Machine

There are 5 virtual machine

Control plane are the master nodes -> 3 nodes

Data plane are the Worker nodes -> 2 nodes

1. AKS (Azure Kubernetes Services)

AKS

Request + Node pool (virtual machine scale set)

1 & 2 are self managed kubernetes

3 azure managed

AKS

Create Kubernetes Cluster ->

Cost –>

Scalability etc

Kubernetes, often abbreviated as K8s, is an open-source platform designed for automating the deployment, scaling, and management of containerized applications.

Here are some key features of Kubernetes:

* **Automated Rollouts and Rollbacks**: Kubernetes can automatically roll out changes to your application or its configuration, and roll them back if something goes wrong.
* **Service Discovery and Load Balancing**: It assigns IP addresses to Pods and provides a single DNS name for a set of Pods, enabling load balancing.
* **Storage Orchestration**: Kubernetes can automatically mount the storage system of your choice, whether from local storage, a public cloud provider, or a network storage system.
* **Self-Healing**: It restarts containers that fail, replaces and reschedules containers when nodes die, and kills containers that don’t respond to user-defined health checks.
* [**Horizontal Scaling**: You can scale your application up and down with a simple command, a UI, or automatically based on CPU usage](https://kubernetes.io/)

# Day - 17 | Deploy an E-Commerce Project on Azure Kubernetes Service | Step By Step Guide Project AKS

(8 services, 2 Databases)

Application consist of 8 microservices -> 2 databases that are involved

MySQL database and MongoDB database

Memory data store

This project is a combination of deployments stateful sets and persistent volumes storage classes along with that we will also implement ingress configuration

1. Deploy the application
2. Create the ingress controller
3. Ingress to expose this application

Architecture of application

<https://github.com/iam-veeramalla/three-tier-architecture-demo>

Mysql -> relational database

NoSql -> non relational database

Terraform

Github workflow

GitHub Actions workflows

A **workflow** is a configurable automated process that will run one or more jobs. Workflows are defined by a YAML file checked in to your repository and will run when triggered by an event in your repository, or they can be triggered manually, or at a defined schedule.

Workflows are defined in the .github/workflows directory in a repository. A repository can have multiple workflows, each which can perform a different set of tasks such as:

* Building and testing pull requests.
* Deploying your application every time a release is created.
* Adding a label whenever a new issue is opened.

1. [**Workflow basics**](https://docs.github.com/en/actions/writing-workflows/about-workflows#workflow-basics)

A workflow must contain the following basic components:

1. One or more *events* that will trigger the workflow.
2. One or more *jobs*, each of which will execute on a *runner* machine and run a series of one or more *steps*.
3. Each step can either run a script that you define or run an action, which is a reusable extension that can simplify your workflow.

* [**Triggering a workflow**](https://docs.github.com/en/actions/writing-workflows/about-workflows#triggering-a-workflow)

Workflow triggers are events that cause a workflow to run. These events can be:

* Events that occur in your workflow's repository
* Events that occur outside of GitHub and trigger a repository\_dispatch event on GitHub
* Scheduled times
* Manual

For example, you can configure your workflow to run when a push is made to the default branch of your repository, when a release is created, or when an issue is opened.

For more information, see "[Triggering a workflow](https://docs.github.com/en/actions/using-workflows/triggering-a-workflow)", and for a full list of events, see "[Events that trigger workflows](https://docs.github.com/en/actions/using-workflows/events-that-trigger-workflows)."

## [Workflow syntax](https://docs.github.com/en/actions/writing-workflows/about-workflows#workflow-syntax)

Workflows are defined using YAML. For the full reference of the YAML syntax for authoring workflows, see "[Workflow syntax for GitHub Actions](https://docs.github.com/en/actions/using-workflows/workflow-syntax-for-github-actions#about-yaml-syntax-for-workflows)."

For more on managing workflow runs, such as re-running, cancelling, or deleting a workflow run, see "[Managing workflow runs and deployments](https://docs.github.com/en/actions/managing-workflow-runs)."

## [Using workflow templates](https://docs.github.com/en/actions/writing-workflows/about-workflows#using-workflow-templates)

GitHub provides preconfigured workflow templates that you can use as-is or customize to create your own workflow. GitHub analyses your code and shows you workflow templates that might be useful for your repository. For example, if your repository contains Node.js code, you'll see suggestions for Node.js projects.

These workflow templates are designed to help you get up and running quickly, offering a range of configurations such as:

* CI: [Continuous Integration workflows](https://github.com/actions/starter-workflows/tree/main/ci)
* Deployments: [Deployment workflows](https://github.com/actions/starter-workflows/tree/main/deployments)
* Automation: [Automating workflows](https://github.com/actions/starter-workflows/tree/main/automation)
* Code Scanning: [Code Scanning workflows](https://github.com/actions/starter-workflows/tree/main/code-scanning)
* Pages: [Pages workflows](https://github.com/actions/starter-workflows/tree/main/pages)

Use these workflows as a starting place to build your custom workflow or use them as-is. You can browse the full list of workflow templates in the [actions/starter-workflows](https://github.com/actions/starter-workflows) repository. For more information, see "[Using workflow templates](https://docs.github.com/en/actions/writing-workflows/using-starter-workflows)."

[**Storing secrets**](https://docs.github.com/en/actions/writing-workflows/about-workflows#storing-secrets)

If your workflows use sensitive data, such as passwords or certificates, you can save these in GitHub as *secrets* and then use them in your workflows as environment variables. This means that you will be able to create and share workflows without having to embed sensitive values directly in the workflow's YAML source.

This example job demonstrates how to reference an existing secret as an environment variable, and send it as a parameter to an example command.

jobs:

example-job:

runs-on: ubuntu-latest

steps:

- name: Retrieve secret

env:

super\_secret: ${{ secrets.SUPERSECRET }}

run: |

example-command "$super\_secret"

[**Creating dependent jobs**](https://docs.github.com/en/actions/writing-workflows/about-workflows#creating-dependent-jobs)

By default, the jobs in your workflow all run in parallel at the same time. If you have a job that must only run after another job has completed, you can use the needs keyword to create this dependency. If one of the jobs fails, all dependent jobs are skipped; however, if you need the jobs to continue, you can define this using the if conditional statement.

In this example, the setup, build, and test jobs run in series, with build and test being dependent on the successful completion of the job that precedes them:

jobs:

setup:

runs-on: ubuntu-latest

steps:

- run: ./setup\_server.sh

build:

needs: setup

runs-on: ubuntu-latest

steps:

- run: ./build\_server.sh

test:

needs: build

runs-on: ubuntu-latest

steps:

- run: ./test\_server.sh

1. [**Using a matrix**](https://docs.github.com/en/actions/writing-workflows/about-workflows#using-a-matrix)

A matrix strategy lets you use variables in a single job definition to automatically create multiple job runs that are based on the combinations of the variables. For example, you can use a matrix strategy to test your code in multiple versions of a language or on multiple operating systems. The matrix is created using the strategy keyword, which receives the build options as an array. For example, this matrix will run the job multiple times, using different versions of Node.js:

jobs:

build:

runs-on: ubuntu-latest

strategy:

matrix:

node: [14, 16]

steps:

- uses: actions/setup-node@v4

with:

node-version: ${{ matrix.node }}

