

Tutorial Module 11: Deployment on Kubernetes **Advanced Programming**



Compiled by: Daya Adianto

Editor: Teaching Team (Lecturer and Teaching Assistant)

Advanced Programming

Semester Genap 2024

Fasilkom UI

Author: *Daya Adianto*

Email: dayaadianto@cs.ui.ac.id

© 2024 Faculty of Computer Science Universitas Indonesia

This work uses license: [Creative Commons Attribution-ShareAlike 4.0 International \(CC BY-SA 4.0\)](https://creativecommons.org/licenses/by-sa/4.0/)

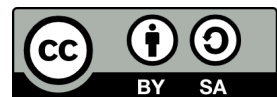


Table of Contents

Table of Contents	1
Learning Objectives	2
References	2
Tutorial	3
Tutorial: Hello Minikube	3
Create a Deployment	4
Create a Service	8
Monitor Resource Usage	10
Reflection on Hello Minikube	13
Tutorial: Rolling Update Deployment	14
Create Another Deployment	14
Create Another Service	15
Scale the App	16
Perform a Rolling Update	18
Write Kubernetes Manifest Files	21
Reflection on Rolling Update & Kubernetes Manifest File	22
Grading Scheme	23
Scale	23
Components	23
Rubrics	23
Expected Individual Implementation to the Group Project	24

Learning Objectives

1. Students are able to use container-based technology for packaging and deploying applications.
2. Students know deployment strategies such as rolling update and recreate
3. Students are able to operate Kubernetes cluster to deploy an application

References

1. Kubernetes documentation: <https://kubernetes.io/docs/home/>
2. Slidedeck about “Container” from Advanced Programming 2023 (KKI)
3. Slidedeck about “Deployment Strategies” from APAP 2023 (IS)

Tutorial

Hello, and welcome to the tutorial module on Kubernetes! You will practice on the basics of using Kubernetes. To complete the tutorial and apply it to your work, you need to prepare the following tools in your local development environment:

- [Docker](#)
- [Minikube](#)
- [`kubectl`](#)

Install the tools above first before you start. Please read the installation instructions carefully. Hopefully there won't be any issues.

Tutorial: Hello Minikube

Let's start the tutorial by running a Minikube cluster locally.

Open a shell and run ``minikube start`` command:

(Warning: the expected, similar output is printed after the command invocation. do not copy-paste the output to your shell)

```
```shell
```

```
$ minikube start
```

```
😊 minikube v1.33.0 on Microsoft Windows 10 Pro 10.0.19045.4291 Build 19045.4291
```

```
✨ Using the docker driver based on existing profile
```

```
👍 Starting "minikube" primary control-plane node in "minikube" cluster
```

```
🚜 Pulling base image v0.0.43 ...
```

```
👤 Updating the running docker "minikube" container ...
```

```
🐳 Preparing Kubernetes v1.30.0 on Docker 26.0.1 ...
```

```
🔍 Verifying Kubernetes components ...
```

- Using image `gcr.io/k8s-minikube/storage-provisioner:v5`

☀ Enabled addons: storage-provisioner, default-storageclass

🏃 Done! `kubectl` is now configured to use "minikube" cluster and "default" namespace by default

...

Then, open a different shell and run ``minikube dashboard``. The command will set up a Web-based dashboard that you can use to manage the locally-running Minikube cluster. Once ``minikube`` finishes preparing the dashboard, it will open the dashboard on a new browser window/tab. **Keep the shell running** throughout the tutorial. You can monitor the cluster through the Web-based dashboard or using ``kubectl`` commands.

## Create a Deployment

Now, let us proceed to deploy a sample container image using ``kubectl`` to the Minikube cluster.

Open a shell and follow these steps:

1. Use the ``kubectl create`` command to create a Deployment that manages a Pod.

The Pod runs a container based on the provided image:

```
""shell

kubectl create deployment hello-node
--image=registry.k8s.io/e2e-test-images/agnhost:2.39 -- /agnhost
netexec --http-port=8080
```

...

2. View the Deployment:

```
""shell

kubectl get deployments

...
```

The output is similar to:

```
""shell
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
hello-node	1/1	1	1	27s

```
""
```

### 3. View the Pod:

```
""shell
```

```
kubectl get pods
```

```
""
```

The output is similar to:

```
""shell
```

NAME	READY	STATUS	RESTARTS	AGE
hello-node-55fdcd95bf-dgz9z	1/1	Running	0	2m23s

```
""
```

### 4. View cluster events:

```
""shell
```

```
kubectl get events
```

```
""
```

The output is similar to:

```
""shell
```

LAST SEEN	TYPE	REASON	OBJECT
MESSAGE			

4m54s	Normal	Scheduled	pod/hello-node-55fdcd95bf-dgz9z Successfully assigned default/hello-node-55fdcd95bf-dgz9z to minikube
-------	--------	-----------	-------------------------------------------------------------------------------------------------------

```

4m53s Normal Pulling
pod/hello-node-55fdcd95bf-dgz9z Pulling image
"registry.k8s.io/e2e-test-images/agnhost:2.39"

4m40s Normal Pulled
pod/hello-node-55fdcd95bf-dgz9z Successfully pulled image
"registry.k8s.io/e2e-test-images/agnhost:2.39" in 13.12s (13.12s
including waiting). Image size: 126872991 bytes.

4m40s Normal Created
pod/hello-node-55fdcd95bf-dgz9z Created container agnhost

4m40s Normal Started
pod/hello-node-55fdcd95bf-dgz9z Started container agnhost

4m54s Normal SuccessfulCreate
replicaset/hello-node-55fdcd95bf Created pod:
hello-node-55fdcd95bf-dgz9z

4m54s Normal ScalingReplicaSet deployment/hello-node
Scaled up replica set hello-node-55fdcd95bf to 1

14m Normal Starting node/minikube
Starting kubelet.

14m Normal NodeAllocatableEnforced node/minikube
Updated Node Allocatable limit across pods

14m Normal NodeHasSufficientMemory node/minikube
Node minikube status is now: NodeHasSufficientMemory

14m Normal NodeHasNoDiskPressure node/minikube
Node minikube status is now: NodeHasNoDiskPressure

14m Normal NodeHasSufficientPID node/minikube
Node minikube status is now: NodeHasSufficientPID

14m Normal RegisteredNode node/minikube
Node minikube event: Registered Node minikube in Controller

13m Normal Starting node/minikube

```

5. View the `kubectl` configuration:

```
""shell

kubectl config view
```

The output is the same as the content of the `config` file located at `\$HOME/.kube/config`.  
(`\$HOME` is your home directory. For Windows users, it is located at `C:\Users\<USER>\`.)

6. View application logs for a container in a pod:

```
""shell

kubectl logs <POD_NAME>
```

The output is similar to:

```
""shell

I0509 21:24:05.853748 1 log.go:195] Started HTTP server on
port 8080

I0509 21:24:05.854115 1 log.go:195] Started UDP server on
port 8081
```

7. Did you encounter any problems in the above steps? We hope not.

But if you did, do not be afraid to ask.



## Create a Service

We have confirmed that the sample application is running in the cluster. But at its current state, we are not able to access it from outside of the cluster. We need to expose the Pod as Service so that they can be accessed by external clients.

Go through the following steps to make the Pod accessible from outside of the cluster:

1. Expose the Pod to the public internet using the `kubectl expose` command:

```
""shell

kubectl expose deployment hello-node --type=LoadBalancer
--port=8080
```

*> Note: The `--type=LoadBalancer` flag is used to create a Service of type LoadBalancer that exposes the Pod to outside of the cluster.*

2. View the Service you created:

```
""shell

kubectl get services
```

The output is similar to:

```
""shell
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
hello-node	LoadBalancer	10.105.254.169	<pending>	8080:31832/TCP
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP

> Note: The `CLUSTER-IP` is an internal IP address that is used by the Pod to communicate with each other.

> The `hello-node` Service has an external IP address (`EXTERNAL-IP`) that can be assigned to a public DNS name or a fixed IP address,

> which requires another configuration step on an actual Kubernetes cluster.

3. Run the following command:

```
""shell


minikube service hello-node

""
```

The output in the terminal is similar to:

```
""shell
```

NAMESPACE	NAME	TARGET PORT	URL
default	hello-node	8080	http://192.168.49.2:31832

 Starting tunnel for service hello-node.

NAMESPACE	NAME	TARGET PORT	URL
default	hello-node		http://127.0.0.1:58518



Opening service default/hello-node in default browser ...

! Because you are using a Docker driver on windows, the terminal needs to be open to run it.

...

Minikube creates a tunnel to the cluster and then runs a proxy to forward traffic from localhost to the service.

Hence, you can access the app via the URL (e.g., <http://127.0.0.1:58518>) in your browser.

Note that the port number might be different in your development environment.

4. To stop the running proxy in the terminal, use CTRL-C (Windows) or CMD-C (macOS).
5. View the application logs again using `kubectl logs` command and take note of the output.

## Monitor Resource Usage

You can monitor resource usage of a cluster by using `kubectl top node` and `kubectl top pod` commands:

```
""shell
```

```
kubectl top nodes
```

```
kubectl top pods
```

...

However, it is likely that you will get the following error message when running the above commands in Minikube:

```
""shell
```

```
error: Metrics API not available
```

...

You need to enable metrics server addon in the cluster, so the resource usage can be captured and monitored.

To do so, follow these steps:

1. Enable `metrics-server` addon:

```
```shell
minikube addons enable metrics-server
```
```

The output is similar to:

```
```shell
The 'metrics-server' addon is enabled
```
```

*> Note: You can also check the list of supported addons by running `minikube addons list`.*

2. Verify the Pod and Service related to the metrics server are created:

```
```shell
kubectl get pods,services -n kube-system
```
```

The output is similar to:

```
```shell

```

	NAME	READY	STATUS	RESTARTS
AGE				
	pod/coredns-7db6d8ff4d-mc82r	1/1	Running	0
58m				
	pod/etcd-minikube	1/1	Running	0
58m				
	pod/kube-apiserver-minikube	1/1	Running	0
58m				

```
```shell

```

```

58m pod/kube-controller-manager-minikube 1/1 Running 0
58m pod/kube-proxy-lc8vc 1/1 Running 0
58m pod/kube-scheduler-minikube 1/1 Running 0
7m25s pod/metrics-server-c59844bb4-scmh7 1/1 Running 0
58m pod/storage-provisioner 1/1 Running 1 (57m ago)

```

NAME PORT(S)	AGE	TYPE	CLUSTER-IP	EXTERNAL-IP
service/kube-dns 53/UDP,53/TCP,9153/TCP	58m	ClusterIP	10.96.0.10	<none>
service/metrics-server 443/TCP	7m25s	ClusterIP	10.100.248.221	<none>

...

> Note: You may need to wait a bit, around 2 - 3 minutes, until `metrics-server` is operational in the cluster.

3. Check the output from `metrics-server` using `kubectl top` commands:

```
```shell
```

```
kubectl top nodes
```

```
kubectl top pods
```

...

The output is similar to:

```
```shell
```

```
kubectl top nodes
```

NAME	CPU(cores)	CPU%	MEMORY(bytes)	MEMORY%
minikube	101m	2%	973Mi	24%

```
kubectl top pods
```

NAME	CPU(cores)	MEMORY(bytes)
hello-node-55fdcd95bf-dgz9z	1m	12Mi

```
```
```

Let `metrics-server` run throughout the tutorial.

If you want to disable it later, you can delete it using:

```
```shell
```

```
minikube addons disable metrics-server
```

```
```
```

Reflection on Hello Minikube

At the end of the "Hello Minikube" tutorial, you have experienced how to deploy and manage Kubernetes resources using Minikube. Now take a breath, grab some snacks, and try to reflect on your experiences.

Answer the following questions to guide your reflection:

1. Compare the application logs before and after you exposed it as a Service.

Try to open the app several times while the proxy into the Service is running.

What do you see in the logs? Does the number of logs increase each time you open the app?

2. Notice that there are two versions of `kubectl get` invocation during this tutorial section.

The first does not have any option, while the latter has `-n`` option with value set to `kube-system``.

What is the purpose of the `-n`` option and why did the output not list the pods/services that you explicitly created?

> *Hint: Do some reading about [Namespace in Kubernetes documentation](<https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/>).*

To make notes of your experience and reflection, please do the following:

- ☐ **Create a new Git repository on GitLab/GitHub and make it available to your TA.**
- ☐ **Write your answers into a file named README.md and store the file into the Git repository.**
- ☐ **Make sure you have committed and push the README.md file to the online Git repository.**

Tutorial: Rolling Update Deployment

Now let us try deploying another app example to Minikube cluster and experiment with [rolling update](#) deployment strategy. This tutorial section is based on the ["Learn Kubernetes Basics" tutorial](#) with minor modifications. You will reuse the Minikube cluster you have used in the previous tutorial section. In addition, instead of deploying the same app mentioned in the original tutorial, you are going to deploy a Spring Boot-based app called [Spring Petclinic REST](#).

Create Another Deployment

To deploy an app to a Kubernetes cluster, remember that the app needs to be packaged into a container image first and then pushed into a container registry. You may remember the deployment step you have done in the PBP course tutorial last semester. You have created a `Dockerfile`` and store it along with your tutorial code. Then, the CI/CD pipeline (i.e., GitHub Actions) pushes your latest commit to PBP's PaaS that will build your app into a container image and store it into a container registry that is hosted locally on the faculty's server.

In this tutorial, you will try deploying another app that has been packaged into a container image and published on a public container registry named [Docker Hub](#). The app you will deploy is called Spring Petclinic REST and you can see the list of published images at the following URL: <https://hub.docker.com/r/springcommunity/spring-petclinic-rest/tags>

1. Since we want to try rolling update deployment, deploy the app with a new tag name `3.0.2` instead of the latest tag name (3.2.1):

```
```shell

 kubectl create deployment spring-petclinic-rest
--image=docker.io/springcommunity/spring-petclinic-rest:3.0.2

```
```

2. Verify the new Deployment and the Pod are created:

```
```shell

 kubectl get deployments

 kubectl get pods

```
```

Make sure the corresponding Deployment and Pod are running and ready.

Then, view the application logs:

```
```shell

 kubectl logs <POD_NAME>

```
```

Take note of the port number listed by the web server (Tomcat).

The port number used by the web server in the example should be **9966**.

Create Another Service

1. Create a new Service for Spring Petclinic REST app:

```
```shell

 kubectl expose deployment/spring-petclinic --type="LoadBalancer"
--port 9966

```
```


2. Verify the new Service is created:

```
```shell
kubectll get services
```
```

3. Open the deployed app via tunnel:

```
```shell
minikube service spring-petclinic-rest
```
```

The browser should show the Spring Petclinic REST app. The REST API can be accessed at `/petclinic`` subpath. You can play around using Postman, ``curl``, or directly on the provided Swagger API that can be accessed via ``http://127.0.0.1:<PORT>/petclinic``

Scale the App

One of the benefits of using Kubernetes is the ability to scale up and down the app. It is accomplished by changing the number of replicas (ReplicaSet) in a Deployment:

1. List the Deployment running in the cluster:

```
```shell
kubectll get deployments
```
```

Look at the ``READY`` column. It represents the ratio of current and desired number of replicas.

2. View the ReplicaSet created by the Deployment:

```
```shell
kubectll get rs
```
```

Two important columns in the output are: `DESIRED` and `CURRENT`.

The `DESIRED` is the **desired** number of replicas of the application, which is defined when creating the Deployment.

The `CURRENT` is the number of replicas that are actually running.

3. Try to scale the Deployment to 4 replicas by using `kubectl scale`:

```
```shell
kubectl scale deployments/spring-petclinic-rest --replicas=4
```
```

4. Verify the Deployment and ReplicaSet are scaled correctly:

```
```shell
kubectl get deployments
kubectl get rs
```
```

The number of application instances should be increased to 4.

5. Verify there are 4 Pods running Spring Petclinic REST:

```
```shell
kubectl get pods -o wide
```
```

The output is similar to:

```
```shell
```

	NAME		READY	STATUS	RESTARTS	AGE
IP	NODE	NOMINATED NODE	READINESS	GATES		
	hello-node-55fdcd95bf-dgz9z		1/1	Running	0	130m
10.244.0.3	minikube	<none>	<none>			

```
```
```

```

    spring-petclinic-7b976f75d4-k8rbv    1/1    Running    0
2m55s   10.244.0.12    minikube    <none>    <none>

    spring-petclinic-7b976f75d4-vrntb    1/1    Running    0
2m55s   10.244.0.11    minikube    <none>    <none>

    spring-petclinic-7b976f75d4-xccdh    1/1    Running    0
17m    10.244.0.9    minikube    <none>    <none>

    spring-petclinic-7b976f75d4-xqxfs    1/1    Running    0
2m55s   10.244.0.10    minikube    <none>    <none>
...

```

Perform a Rolling Update

A **rolling update** allows an app to be updated without downtime. It is done by incrementally replacing the current Pods with new ones, which are running the new version of the app. The new Pods are scheduled on Nodes with available resources. Once all new Pods are started, the old Pods will be removed.

Let's update the Spring Petclinic REST app by replacing the image with the newer version:

1. Check the current deployment specification and see if the image is using the old version (3.0.2):

```

```shell

kubectl describe deployment/spring-petclinic-rest
...

```

Look output similar to:

```

```shell

Pod Template:

Labels:  app=spring-petclinic

Containers:
  spring-petclinic-rest:

```

```
Image:      docker.io/springcommunity/spring-petclinic-rest:3.0.2
```

```
'''
```

2. Update the image to use the newer version (in this case `3.2.1`):

```
'''shell
```

```
kubectl set image deployments/spring-petclinic-rest
spring-petclinic-rest=docker.io/springcommunity/spring-petclinic-rest:
3.2.1
```

```
'''
```

The command notifies the Deployment to use a different image for the app and starts a rolling update. You can check the status of the new Pods and view the old ones terminating with the `kubectl get pods` command.

3. Verify the update status by running `kubectl rollout status`:

```
'''shell
```

```
kubectl rollout status deployments/spring-petclinic-rest
```

```
'''
```

4. Re-check the deployment specification and the pods to see if the image is using the new version:

```
'''shell
```

```
kubectl describe deployment/spring-petclinic-rest
```

```
'''
```

You should see output that contains the similar text as follows:

```
'''shell
```

```
[ ... ]
```

```
Containers:
```

```
spring-petclinic-rest:
```

```
Image:      docker.io/springcommunity/spring-petclinic-rest:3.2.1
[ ... ]

OldReplicaSets:  spring-petclinic-7b976f75d4 (0/0 replicas
created)

NewReplicaSet:   spring-petclinic-5d75597599 (4/4 replicas
created)

'''
```

5. Now let's try to update the deployed image to a non-existing version.

```
'''shell

kubectl set image deployments/spring-petclinic-rest
spring-petclinic-rest=docker.io/springcommunity/spring-petclinic-rest:
4.0

'''
```

6. See the status of Deployment:

```
'''shell

kubectl get deployments

kubectl get pods

kubectl describe pods

'''
```


Notice that some of the Pods have a status of `ImagePullBackOff`. In addition, the Events section of the affected Pods notified the `4.0` image version did not exist.

7. Roll back the deployment to the last working version using `kubectl rollout undo`:

```
'''shell

kubectl rollout undo deployments/spring-petclinic-rest

'''
```



The `rollout undo` command reverts the deployment to the previous known state. The updates are versioned and you can revert to any previously known state of a Deployment.

8. Verify that the Deployment has been reverted to use the previously working image (3.2.1):

```
```shell
kubectrl get pods
kubectrl describe pods
```
```

Write Kubernetes Manifest Files

By the time you reach this subsection, you have experienced deploying an app to Kubernetes *programmatically* by using `kubectrl`. It is also possible to deploy an app to Kubernetes using a YAML file (Kubernetes manifest file) that is read by `kubectrl`.

Let's start by exporting the existing configuration to YAML files:

1. Export the deployment configuration:

```
```shell
kubectrl get deployments/spring-petclinic-rest -o yaml >
deployment.yaml
```
```

2. Export the service configuration:

```
```shell
kubectrl get services/spring-petclinic-rest -o yaml > service.yaml
```
```

3. You can try to reset/destroy the Minikube cluster and start over again from a blank state.

```
```shell
minikube delete
```
```

```
minikube start
```

```
'''
```

4. Then, apply the Kubernetes manifest files you have created to set up your deployment again in the new cluster.

```
'''shell
```

```
kubectl apply -f deployment.yaml
```

```
kubectl apply -f service.yaml
```

```
'''
```

5. Save the manifest files as a new Git commit and push the new commit to your online Git repository.

Reflection on Rolling Update & Kubernetes Manifest File

You have reached the end of the tutorial. By now, you have experienced how to perform rolling updates on Kubernetes and write some Kubernetes manifest files.

Try to reflect on your experiences again and answer the following questions:

1. What is the difference between Rolling Update and Recreate deployment strategy?

> *Hint: Read the [Deployments documentation](#).*

2. Try deploying the Spring Petclinic REST using Recreate deployment strategy and document your attempt.

3. Prepare different manifest files for executing Recreate deployment strategy.

4. What do you think are the benefits of using Kubernetes manifest files? Recall your experience in deploying the app manually and compare it to your experience when deploying the same app by applying the manifest files (i.e., invoking `kubectl apply -f` command) to the cluster.

5. (Optional) Do the same tutorial steps, but on a managed Kubernetes cluster (e.g., GCP). You need to provision a Kubernetes cluster on Google Cloud Platform. Then, re-run the tutorial steps (Hello Minikube and Rolling Update) on the remote cluster. Document your attempt and highlight the differences and any issues you encountered.

☐ **Write your answers into the same `README.md` from previous reflection and save it as a new Git commit.**

- ☐ **Make sure the Kubernetes manifest files are in the same directory with the ``README.md``.**
- ☐ **Then, push the new commit to your online Git repository.**

Grading Scheme

Scale

All components will be scored on scale 0-4. Grade 4 is the highest grade, equivalent to A. Score 0 is for not submitting.

Components

- 50% - Commits
- 40% - Reflection/Notes Explanation
- 10% - Correctness
- Bonus 10% (Do the same tutorial steps on a managed k8s cluster):
 - 2% - Able to provision the cluster (demonstrated during claim/demo session),
 - 3% - Able to run the app examples on the cluster (demonstrated during claim/demo session),
 - 5% - Detailed explanation and opinion in the reflection notes

Rubrics

| | Score 4 | Score 3 | Score 2 | Score 1 |
|------------------------------|--|---|--|--|
| Correctness | Manifest files can be deployed immediately to Minikube on TA's machine | Require minor changes to the manifest files in order to be deployed to Minikube on TA's machine | Require major changes to the manifest files in order to be deployed to Minikube on TA's machine. | Missing manifest files |
| Reflection/Notes Explanation | More than 5 sentences. The description is sound. | Less than or equal 5 sentences. The description is sound. | The description is not sound although still related. | The description is not sound. It is not related to the topics. |
| Commit | All requested commits are registered and correct. | At least 50% of the requested commits are registered and correct. | At least 25% of the requested commits are registered and correct. | Less than 25%. |

Expected Individual Implementation to the Group Project

Each group member should be able to demonstrate the following to the TA during demo/claim session:

1. Prepare (provision) their own deployment environment
2. Build the container image(s) of the group project
3. Push the container image(s) to an image repository **in their own account/namespace**
4. Deploy the pushed container images to **their own deployment environment** (i.e., not the deployment environment used by the group project or the one that prepared by the group member whose focused on DevOps)

If the group project did not use containers, then each group member should be able to demonstrate to deploy their app to their own deployment environment. For example, if the project uses an IaaS-based approach, then each group member should be able to demonstrate that they can deploy their app to a VM that has been individually provisioned by themselves.