

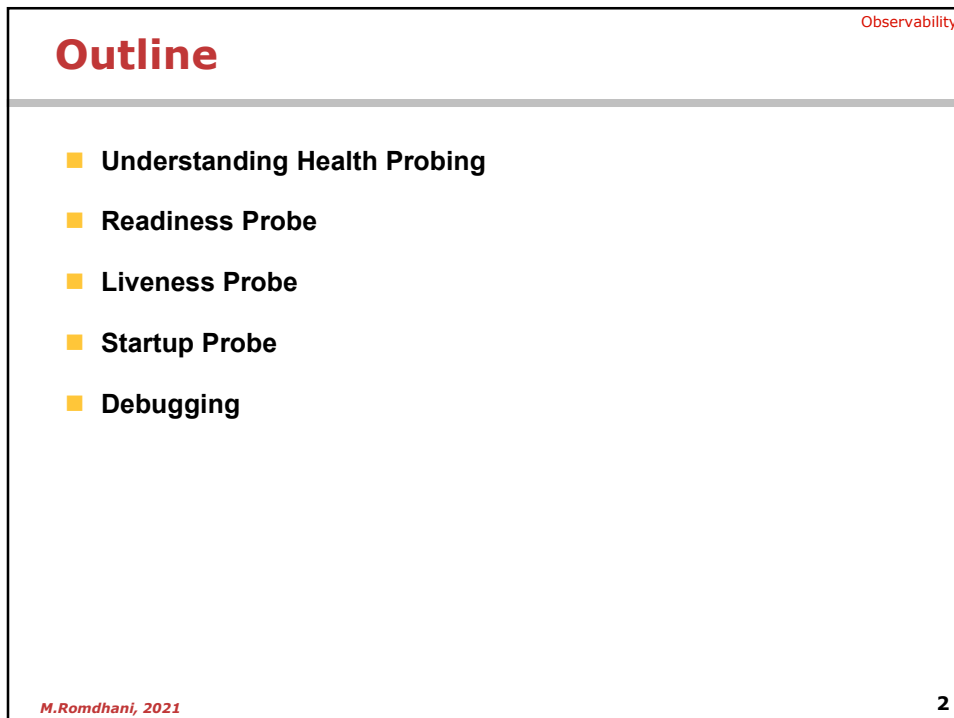
The slide features a purple header and footer with a collage of images including a hand, a person, and a globe. In the center, there is a blue ship's wheel icon above the text "Unit 2". The main title "Healthchecks and Observability" is written in a large, bold, red font. In the bottom right corner, there are three small icons (a circle, a square, and a triangle) above the text "Business Training".

Unit 2

Healthchecks and Observability

Business Training

1



The slide has a white background with a red header bar. The word "Outline" is written in a large, bold, red font. To the right of the header bar, the word "Observability" is written in a smaller, red font. Below the header, there is a list of five items, each preceded by a yellow square bullet point. At the bottom left, the text "M. Romdhani, 2021" is written in a small, red font. At the bottom right, the number "2" is written in a small, red font.

Outline

Observability

- Understanding Health Probing
- Readiness Probe
- Liveness Probe
- Startup Probe
- Debugging

M. Romdhani, 2021

2

2

Understanding Health Probing

3

Why Health probing ?

Observability

- The goal of the observability concern is to be able to know if the application is ready for consumption and is still working as expected in an hour, a week, or a month.
 - Even with the best automated test coverage, it's nearly impossible to find all bugs before deploying software to a production environment.
 - That's especially true for failure situations that only occur after operating the software for an extended period of time.
 - It's not uncommon to see memory leaks, deadlocks, infinite loops, and similar conditions crop up once the application has been put under load by end users.
 - Kubernetes provides a concept called health probing to automate the detection and correction of such issues.
- Types of Healthcheck Probes:**
- Readiness Probes
 - Liveness Probes
 - Startup Probe

M.Romdhani, 2021

4

4

Available health verification methods

Observability

Method	Option	Description
Custom command	exec.command	Executes a command inside of the container (e.g., a cat command) and checks its exit code. Kubernetes considers a zero exit code to be successful. A non-zero exit code indicates an error.
HTTP GET request	httpGet	Sends an HTTP GET request to an endpoint exposed by the application. An HTTP response code in the range of 200 and 399 indicates success. Any other response code is regarded as an error.
TCP socket connection	tcpSocket	Tries to open a TCP socket connection to a port. If the connection could be established, the probing attempt was successful. The inability to connect is accounted for as an error.

M.Romdhani, 2021

5

5

Attributes for fine-tuning the health check runtime behavior

Observability

Attribute	Default value	Description
initialDelaySeconds	0	Delay in seconds until first check is executed.
periodSeconds	10	Interval for executing a check (e.g., every 20 seconds).
timeoutSeconds	1	Maximum number of seconds until check operation times out.
successThreshold	1	Number of successful check attempts until probe is considered successful after a failure.
failureThreshold	3	Number of failures for check attempts before probe is marked failed and takes action.

M.Romdhani, 2021

6

6

Readiness Probe

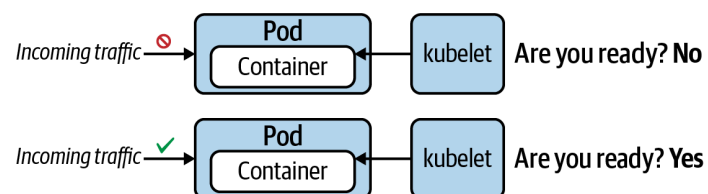
7

Readiness Probes

Observability

■ Readiness probe

- This probe checks if the application is ready to serve incoming requests.



M.Romdhani, 2021

8

8

Readiness Probe Manifest

■ In this example, the readiness probe executes its first check after two seconds and repeats checking every eight seconds thereafter.

- All other attributes use the default values.
- A readiness probe will continue to periodically check, even after the application has been successfully started

```
apiVersion: v1
kind: Pod
metadata:
  name: readiness-pod
spec:
  containers:
    - image: bmuschko/nodejs-hello-world:1.0.0
      name: hello-world
      ports:
        - name: nodejs-port
          containerPort: 3000
      readinessProbe:
        httpGet:
          path: /
          port: nodejs-port
        initialDelaySeconds: 2
        periodSeconds: 8
```

Liveness Probe

Observability

Liveness Probes

- **Liveness probe**
 - This probe checks the application responsiveness
 - Kubernetes restarts the Pod automatically if the probe considers the application be in an unhealthy state

11

M.Romdhani, 2021

11

Observability

Liveness Probe example

- **The probe will periodically check if the modification timestamp of the file is older than one minute.**
 - If it is, then Kubernetes can assume that the application isn't functioning as expected and will restart the container.

```

apiVersion: v1
kind: Pod
metadata:
  name: liveness-pod
spec:
  containers:
  - image: busybox
    name: app
    args:
    - /bin/sh
    - -c
    - 'while true; do touch /tmp/heartbeat.txt; sleep 5; done;'
    livenessProbe:
      exec:
        command:
        - test `find /tmp/heartbeat.txt -mmin -1`
      initialDelaySeconds: 5
      periodSeconds: 30
  
```

12

M.Romdhani, 2021

12

Startup Probe

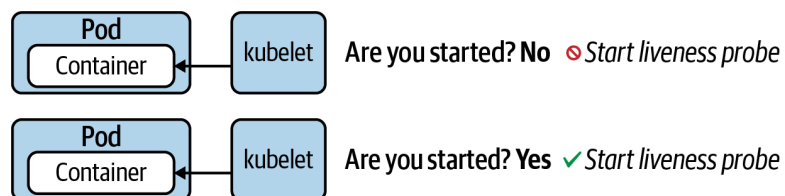
13

Startup Probes

Observability

■ Startup probe

- The purpose of a startup probe is to figure out when an application is fully started. Defining the probe is especially useful for an application that takes a long time to start up.
 - Legacy applications in particular can take a long time to start up—we're talking minutes sometimes.



M.Romdhani, 2021

14

14

Observability

Startup Probe Example

- The Example defines a Pod that runs the Apache HTTP server in a container. By default, the image exposes the container port 80, and that's what we're probing for using a TCP socket

```
apiVersion: v1
kind: Pod
metadata:
  name: startup-pod
spec:
  containers:
  - image: httpd:2.4.46
    name: http-server
    startupProbe:
      tcpSocket:
        port: 80
      initialDelaySeconds: 3
      periodSeconds: 15
```

M.Romdhani, 2021
15

15

Observability

What if I don't specify a liveness probe?

- If you don't specify a liveness probe, then Kubernetes will decide whether to restart your container based on the status of the container's PID 1 process.
 - The PID 1 process is the parent process of all other processes that run inside the container. Because each container begins life with its own process namespace, the first process in the container will assume the special duties of PID 1.
 - If the PID 1 process exits and no liveness probe is defined, Kubernetes assumes (usually safely) that the container has died. Restarting the process is the only application-agnostic, universally effective corrective action. As long as PID 1 is alive, regardless of whether any child processes are running, OpenShift will leave the container running.

M.Romdhani, 2021
16

16

Debugging Pods

17

Troubleshooting Pods

Observability

■ Pod creation problems

- In most cases, creating a Pod is no issue. You simply emit the run, create, or apply commands to instantiate the Pod.
 - If the YAML manifest is formed properly, Kubernetes accepts your request.
- To verify the correct behavior, the first thing you'll want to do is to check the high-level runtime information of the Pod.
 - run either the `kubectl get pods` command for just the Pods running in the namespace or the `kubectl get all` command to retrieve the most prominent object types in the namespace (which includes Deployments).

■ Debugging YAML manifests

- Any incorrect indentation, spelling issue, attribute name, or enumeration will cause a problem during object creation.
- VS Code Yaml Extensions can Help detecting YAML problems
- you might find the browser-based application [Kube YAML](#) helpful. You can simply copy and paste the YAML manifest as text and receive feedback about its correctness.

M.Romdhani, 2021

18

18

Common Pod Error statuses

Status	Root cause	Potential fix
ImagePullBackOff or ErrImagePull	Image could not be pulled from registry.	Check correct image name, check that image name exists in registry, verify network access from node to registry, ensure proper authentication.
CrashLoopBackOff	Application or command run in container crashes.	Check command executed in container, ensure that image can properly execute (e.g., by creating a container with Docker).
CreateContainerConfigError	ConfigMap or Secret referenced by container cannot be found.	Check correct name of the configuration object, verify the existence of the configuration object in the namespace

M.Romdhani, 2021

19

19

Inspecting events

- It's totally possible that you'll not encounter any of those error statuses. But there's still a chance of the Pod having a configuration issue.

- You can retrieve detailed information about the Pod and its events using the `kubectl describe pod` command to inspect its events.
- The following output belongs to a Pod that tries to mount a Secret that doesn't exist. Instead of rendering a specific error message, the Pod gets stuck with the status `ContainerCreating`:

```
$ kubectl get pods
NAME          READY   STATUS             RESTARTS   AGE
secret-pod    0/1     ContainerCreating   0           4m57s
$ kubectl describe pod secret-pod
...
Events:
Type      Reason      Age           From          Message
----      -
Normal    Scheduled   <unknown>     default-scheduler Successfully assigned \
default/secret-pod to kubelet, minikube Unable to attach or \
mount volumes:unmounted volumes=[mysecret], unattached volumes= \
[default-token-bf8rh mysecret]: timed out waiting for condition
```

M.Romdhani, 2021

20

20

Inspecting logs

- When debugging a Pod, the next level of details can be retrieved by downloading and inspecting its logs. You may or may not find additional information that points to the root cause of a misbehaving Pod. It's definitely worth a look. The YAML manifest shown in Example 5-4 defines a Pod running a shell command.

```
$ kubectl logs incorrect-cmd-pod
/bin/sh: unknown: not found
```

- After creating the object, the Pod fails with the status **CrashLoopBackOff**. Running the logs command reveals that the command run in the container has an issue.

- The logs command provides two helpful options I'd like to mention here.

1. The option `-f` streams the logs, meaning you'll see new log entries as they're being produced in real time.
2. The option `--previous` gets the logs from the previous instantiation of a container, which is helpful if the container has been restarted.

Opening an Interactive Shell

- If any of the previous commands don't point you to the root cause of the failing Pod, it's time to open an interactive shell to a container.

- You'll probably know best what behavior to expect from the application at runtime. Ensure that the correct configuration has been created and inspect the running processes.

```
$ kubectl exec failing-pod -it -- /bin/sh
# mkdir -p ~/tmp
# cd ~/tmp
# ls -l
total 4
-rw-r--r- 1          root root 112 May 9 23:52 curr-date.txt
```

Using an Ephemeral Container

- **Some images run in a container are designed to be very minimalistic.**
 - For example, the Google distroless images don't have any Unix utility tools preinstalled. You can't even open a shell to a container, as it doesn't even come with a shell. That's the case for the image `k8s.gcr.io/pause:3.1`, a minimal, distroless image that keeps the container running
- **Kubernetes offers the concept of *ephemeral containers*. Those containers are meant to be disposable and can be deployed for troubleshooting minimal containers that would usually not allow the usage of the `exec` command.**
 - Ephemeral containers are still considered an experimental feature. You will have to explicitly enable the feature with the feature flag `--feature-gates` when starting up the Kubernetes cluster. For example, starting Minikube with this feature can be achieved by using the command `minikube start --feature-gates=EphemeralContainers=true`.

Using an Ephemeral Container

- **Kubernetes 1.18 introduced a new *debug* command that can add an ephemeral container to a running Pod for debugging purposes.**
 - The following command adds the ephemeral container running the image `busybox` to the Pod named `minimal-pod` and opens an interactive shell for it:


```
$ kubectl alpha debug -it minimal-pod --image=busybox
Defaulting debug container name to debugger-jf98g.
If you don't see a command prompt, try pressing enter.
/ # pwd
/
/ # exit
Session ended, resume using 'kubectl alpha attach minimal-pod -c \
debugger-jf98g -i -t' command when the pod is running
```

Debugging Services

25

Troubleshooting Services

Observability

- In case you can't reach the Pods that should map to the Service, start by ensuring that the label selector matches with the assigned labels of the Pods.

- You can query the information by describing the Service and then render the labels of the available Pods with the **option --show-labels**. The following example does not have matching labels and therefore wouldn't apply to any of the Pods running in the namespace:

```
$ kubectl describe service myservice
...
Selector:          app=myapp
...
$ kubectl get pods --show-labels
NAME                                READY   STATUS    RESTARTS   AGE   LABELS
myapp-68bf896d89-qfhlv             1/1     Running   0           7m39s   app=hello
myapp-68bf896d89-tzt55             1/1     Running   0           7m37s   app=world
```

M.Romdhani, 2021

26

26

Observability

Troubleshooting Services

- Alternatively, you can also **query the endpoints** of the Service instance. Say you expected three Pods to be selected by a matching label but only two have been exposed by the Service. You'll want to look at the label selection criteria:

```
$ kubectl get endpoints myservice
```

NAME	ENDPOINTS	AGE
myservice	172.17.0.5:80,172.17.0.6:80	9m31s

M.Romdhani, 2021
27

27

Observability

Troubleshooting Services

- A common source of confusion is the type of a Service. By default, the Service type is ClusterIP, which means that a Pod can only be reached through the Service if queried from the same node inside of the cluster. First, check the Service type.
- If you think that ClusterIP is the proper type you wanted to assign, open an interactive shell from a temporary Pod inside the cluster and run a curl or wget command:

```
$ kubectl get services
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
myservice	ClusterIP	10.99.155.165	<none>	80/TCP	15m

```
$ kubectl run tmp --image=busybox -it --rm -- wget -O- 10.99.155.165:80
```

M.Romdhani, 2021
28

28

Troubleshooting services

- Finally, check if the port mapping from the target port of the Service to the container port of the Pod is configured correctly. Both ports need to match:

```
$ kubectl get service myapp -o yaml | grep targetPort:
  targetPort: 80
$ kubectl get pods myapp-68bf896d89-qfhlv -o yaml | grep containerPort:
  - containerPort: 80
```

Monitoring

- In the Kubernetes world, monitoring tools like Prometheus and Datadog help with collecting, processing, and visualizing the information over time.
- This responsibility falls into the hands of the **metrics server**, a cluster-wide aggregator of resource usage data. Refer to the documentation for more information on its installation process. If you're using Minikube as your practice environment, enabling the metrics-server add-on is straightforward using the following command:

```
$ minikube addons enable metrics-server
The 'metrics-server' addon is enabled
You can now query for metrics of cluster nodes and Pods with the top command:
```

```
$ kubectl top nodes
NAME          CPU(cores)   CPU%   MEMORY(bytes)  MEMORY%
minikube      283m         14%    1262Mi         32%

$ kubectl top pod frontend
NAME          CPU(cores)   MEMORY(bytes)
frontend      0m           2Mi
```

Monitoring

■ Installing Metrics Server on Kubernetes Docker Desktop.

- We need to add the `--kubelet-insecure-tls` argument to the metrics-server deployment, otherwise you'll see an error saying something like unable to fetch metrics from node docker-desktop.

■ Using Helm

```
$ helm install my-metrics-server --set "args={--kubelet-insecure-tls}"
stable/metrics-server --namespace kube-system
```

■ Using Kubectl

- `kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/download/v0.4.2/components.yaml`

The following command patches the deployment:

- `kubectl patch deployment metrics-server -n kube-system --type 'json' -p '[{"op": "add", "path": "/spec/template/spec/containers/0/args/-", "value": "--kubelet-insecure-tls"}]'`