# JChampionConf, 20.01.2023

# Kubernetes Developer Survival Kit


## Sandra Parsick

@SandraParsick
@sparsick@mastodon.social
mail@sandra-parsick.de

# About me

- Sandra Parsick
- Freelance Software Developer and Consultant
- Focus:
  - Java Enterprise
  - Agile Methods
  - Software Craftmanship
  - Automation of Development Process
- Trainings
- Workshops


mail@sandra-parsick.de

@SandraParsick

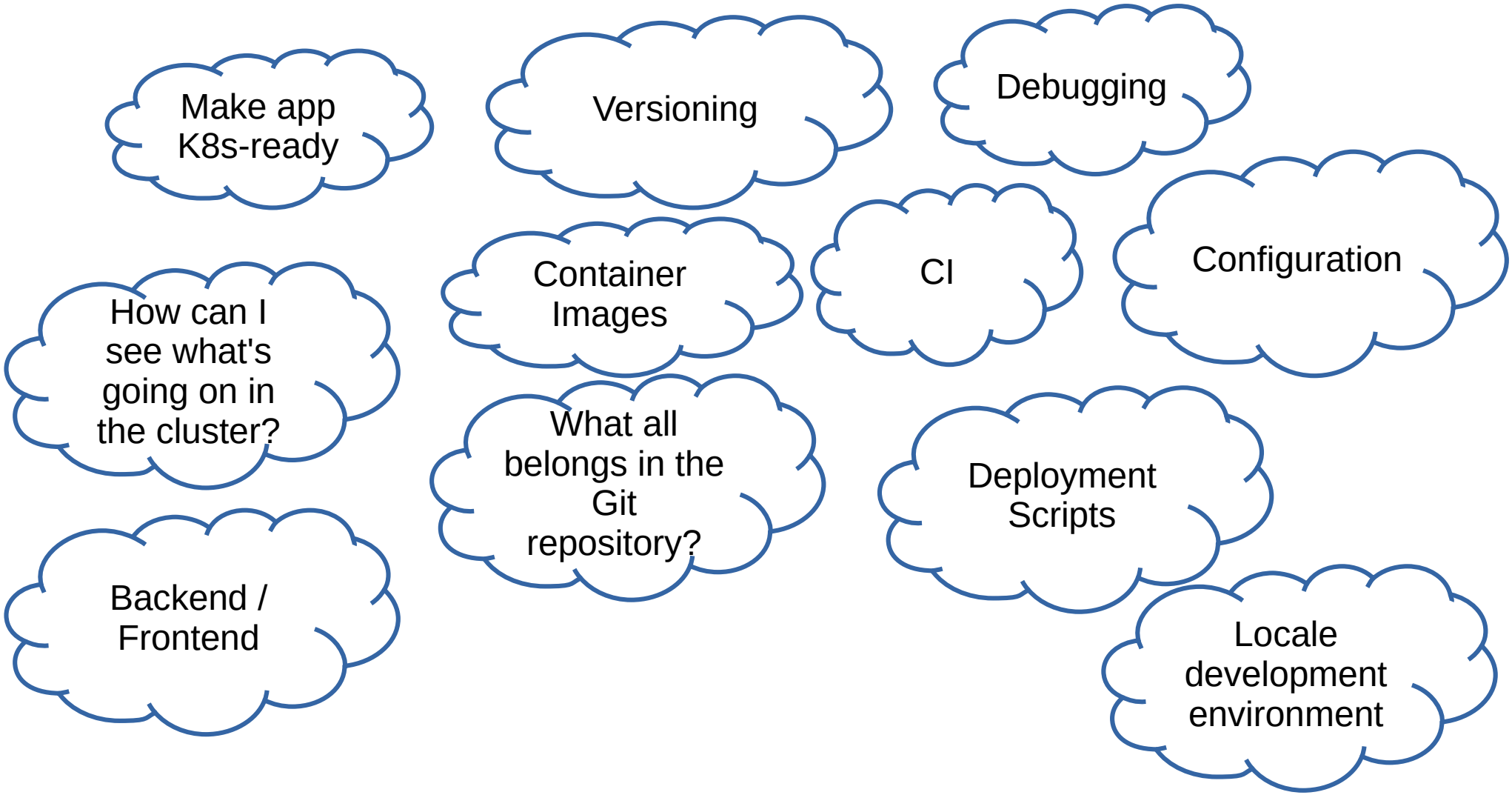
@sparsick@mastodon.social

https://www.sandra-parsick.de

https://ready-for-review.dev




SSA TERMINALS
SSATerminals
NORTHERN JUSTICE
MADEIRA


Make app K8s-ready
Versioning
Debugging
Container Images
CI
Configuration
How can I see what's going on in the cluster?
What all belongs in the Git repository?
Deployment Scripts
Backend / Frontend
Locale development environment

# Friendly Reminder: 12 Factor App

**I. Codebase**
One codebase tracked in revision control, many deploys

**II. Dependencies**
Explicitly declare and isolate dependencies

**III. Config**
Store config in the environment

**IV. Backing services**
Treat backing services as attached resources

**V. Build, release, run**
Strictly separate build and run stages

**VI. Processes**
Execute the app as one or more stateless processes

# Friendly Reminder: 12 Factor App

**VII. Port binding**
Export services via port binding

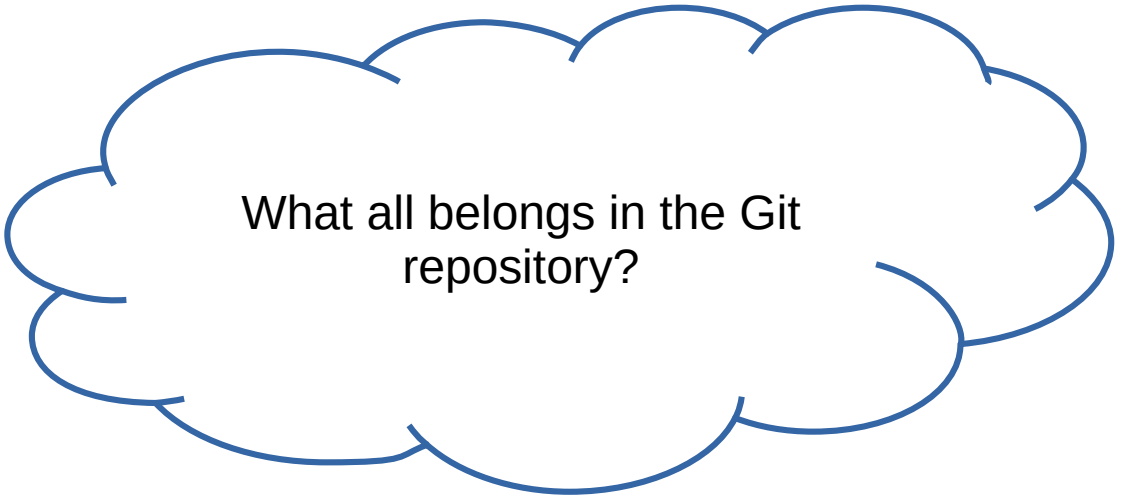
**VIII. Concurrency**
Scale out via the process model

**IX. Disposability**
Maximize robustness with fast startup and graceful shutdown

**X. Dev/prod parity**
Keep development, staging, and production as similar as possible

**XI. Logs**
Treat logs as event streams

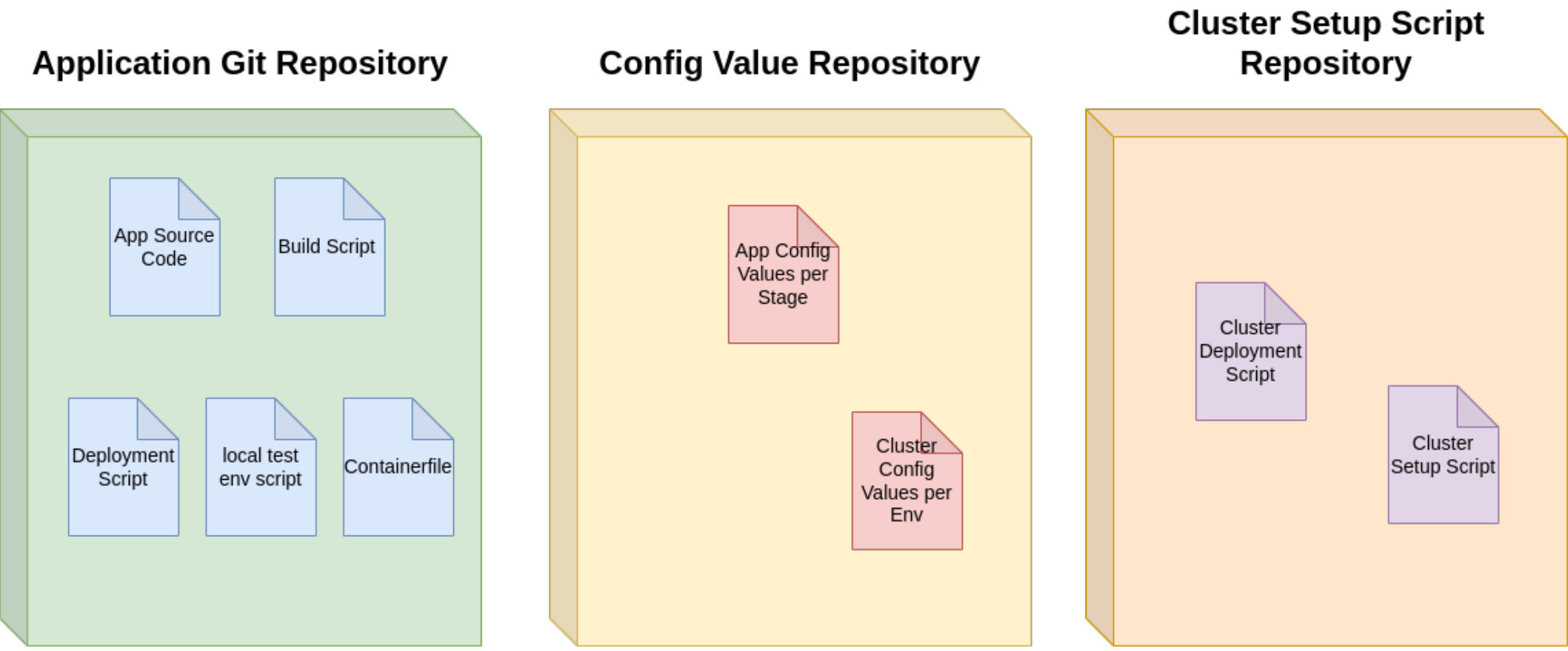
**XII. Admin processes**
Run admin/management tasks as one-off processes


What all belongs in the Git repository?

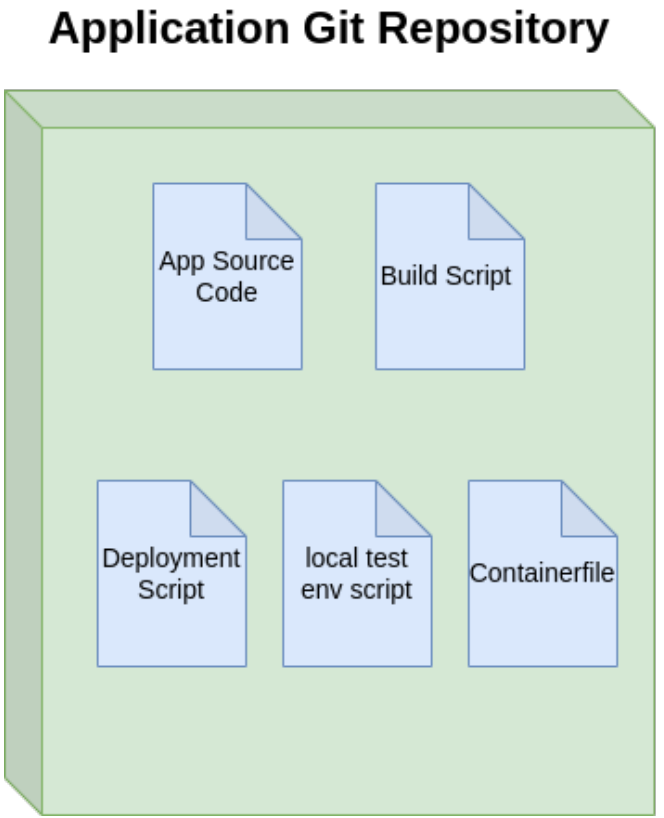
Short form: EVERYTHING

Actual question:
How many repositories?

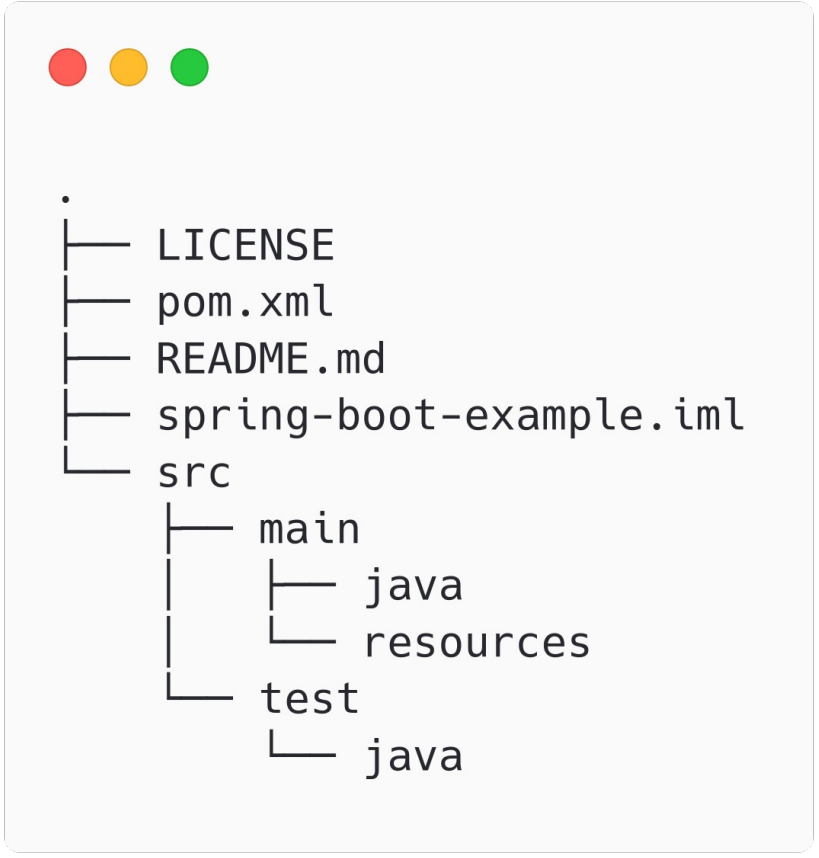
# Example of a split



# Most important for devs

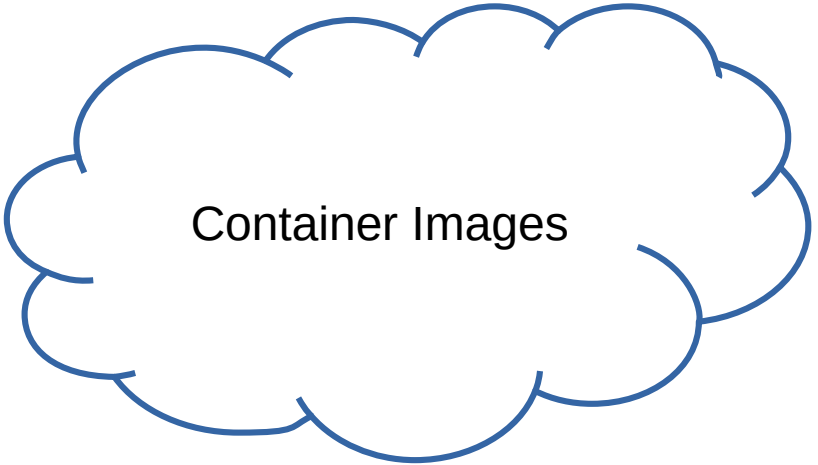


# Starting Point: A Java App

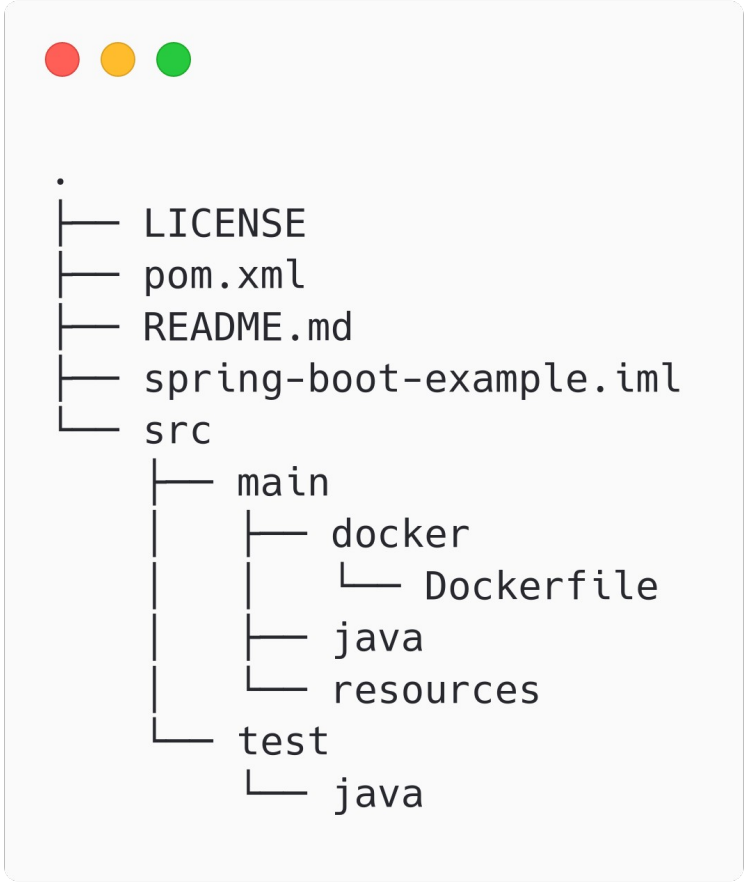



Technology stack:

- Java 17
- Spring Boot 2.7.x
- Thymeleaf
- Apache Maven


Container Images

# Basis: Container



# Basis: Container

```
FROM docker.io/eclipse-temurin:17.0.1_12-jre as builder
WORKDIR /application
COPY maven/*.jar application.jar
RUN java -Djarmode=layertools -jar application.jar extract

FROM gcr.io/distroless/java17-debian11
WORKDIR /application
EXPOSE 8080
COPY --from=builder /application/dependencies/ ./
COPY --from=builder /application/spring-boot-loader/ ./
COPY --from=builder /application/snapshot-dependencies/ ./
COPY --from=builder /application/application/ ./
ENTRYPOINT ["java", "org.springframework.boot.loader.JarLauncher"]
```

```
<plugin>
    <groupId>io.fabric8</groupId>
    <artifactId>docker-maven-plugin</artifactId>
    <version>0.40.0</version>
    <executions>
        <execution>
            <id>docker-build</id>
            <goals>
                <goal>build</goal>
                <goal>push</goal>
            </goals>
        </execution>
    </executions>
    <configuration>
        <images>
            <image>
                <name>spring-boot-demo:latest</name>
                <build>
                    <dockerFile>Dockerfile</dockerFile>
                    <assembly>
                        <descriptorRef>artifact</descriptorRef>
                    </assembly>
                </build>
            </image>
        </images>
        <pushRegistry>localhost:6000</pushRegistry>
  </configuration>
</plugin>
```

# Alternatives

- Buildpacks (spring-boot-maven-plugin)
- JIB (jib-maven-plugin)
- Buildah
- Podman

Container image building is part of the build process and executable locally

# Good Practises Container Image Build

- remove unnecessary tools from the image
- package only one service per image
- build small image
- optimize build cache

- Use own container registry
- Use tags only once when releasing

- Vulnerability-Scans for container images

# Optimize Container Image

```
FROM docker.io/eclipse-temurin:17.0.1_12-jre as builder
WORKDIR /application
COPY maven/*.jar application.jar
RUN java -Djarmode=layertools -jar application.jar extract

FROM gcr.io/distroless/java17-debian11
WORKDIR /application
EXPOSE 8080
COPY --from=builder /application/dependencies/ ./
COPY --from=builder /application/spring-boot-loader/ ./
COPY --from=builder /application/snapshot-dependencies/ ./
COPY --from=builder /application/application/ ./
ENTRYPOINT ["java", "org.springframework.boot.loader.JarLauncher"]
```

# Container Registry

- Cloud Provider:
  - Azure Container Registry
  - AWS Elastic Container Registry
  - Google Container Registry
- On Premise:
  - JFrog Container Registry
  - Red Hat Quay
  - Harbor
  - Artifactory
  - Sonatype Nexus

# Vulnerability-Scans (Ex.: Trivy)

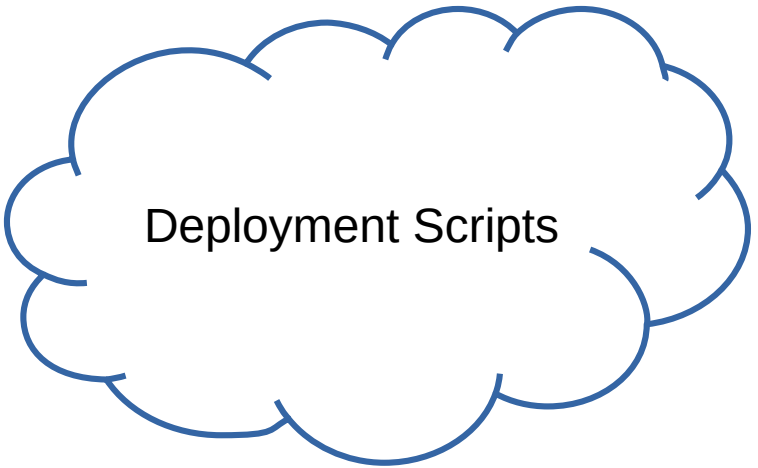
```
→ trivy i --ignore-unfixed -o result  spring-boot-demo:latest
2022-06-23T09:55:56.244+0200    INFO    Vulnerability scanning is enabled
2022-06-23T09:55:56.245+0200    INFO    Secret scanning is enabled
2022-06-23T09:55:56.245+0200    INFO    If your scanning is slow, please try '--security-checks vuln'
to disable secret scanning
2022-06-23T09:55:56.245+0200    INFO    Please see also https://aquasecurity.github.io/trivy/v0.29.2
/docs/secret/scanning/#recommendation for faster secret detection
2022-06-23T09:55:56.254+0200    INFO    Detected OS: debian
2022-06-23T09:55:56.255+0200    INFO    Detecting Debian vulnerabilities...
2022-06-23T09:55:56.265+0200    INFO    Number of language-specific files: 1
2022-06-23T09:55:56.265+0200    INFO    Detecting jar vulnerabilities...

spring-boot-demo:latest (debian 11.2)
=====================================
Total: 23 (UNKNOWN: 1, LOW: 2, MEDIUM: 6, HIGH: 6, CRITICAL: 8)


Java (jar)
==========
Total: 0 (UNKNOWN: 0, LOW: 0, MEDIUM: 0, HIGH: 0, CRITICAL: 0)
```

# Vulnerability-Scans (Ex.: Trivy)

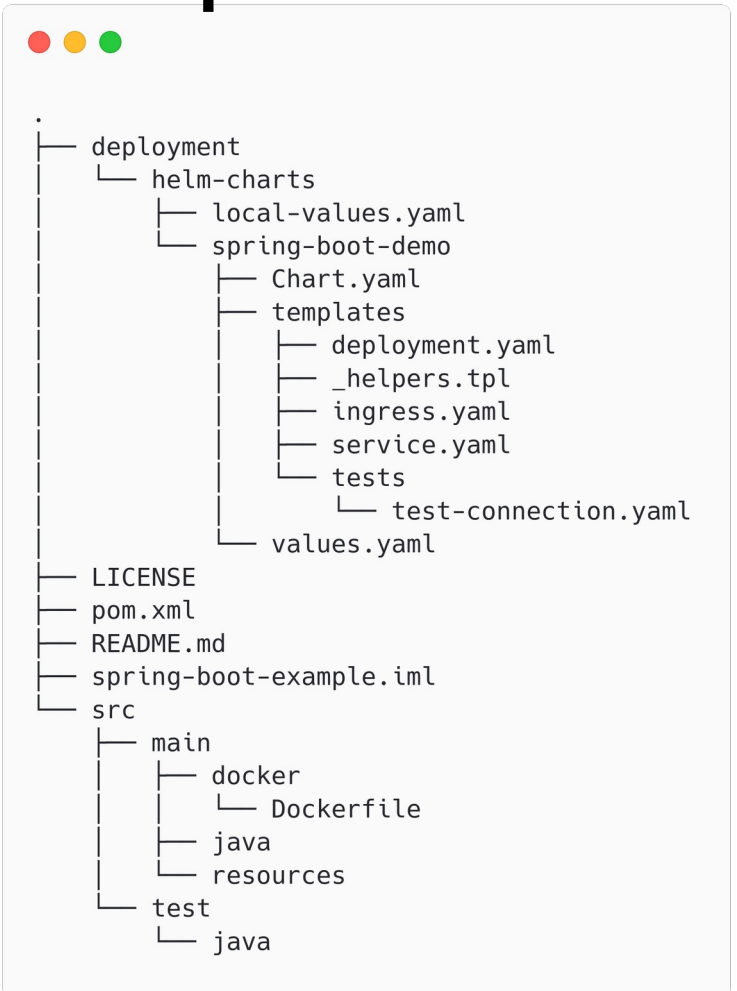
| Library | Vulnerability | Severity | Installed Version | Fixed Version | Title |
|---|---|---|---|---|---|
| libc6 | CVE-2021-33574 | CRITICAL | 2.31-13+deb11u2 | 2.31-13+deb11u3 | glibc: mq_notify does not handle separately allocated thread attributes<br>https://avd.aquasec.com/nvd/cve-2021-33574 |
| | CVE-2022-23218 | | | | glibc: Stack-based buffer overflow in svcunix_create via long pathnames<br>https://avd.aquasec.com/nvd/cve-2022-23218 |
| | CVE-2022-23219 | | | | glibc: Stack-based buffer overflow in sunrpc clnt_create via a long pathname<br>https://avd.aquasec.com/nvd/cve-2022-23219 |
| | CVE-2021-43396 | LOW | | | glibc: conversion from ISO-2022-JP-3 with iconv may emit spurious NUL character on...<br>https://avd.aquasec.com/nvd/cve-2021-43396 |

# Vulnerability-Scans Further Thoughts

- What about
  - containers in registry?
  - containers, that already run in a cluster?


Deployment Scripts

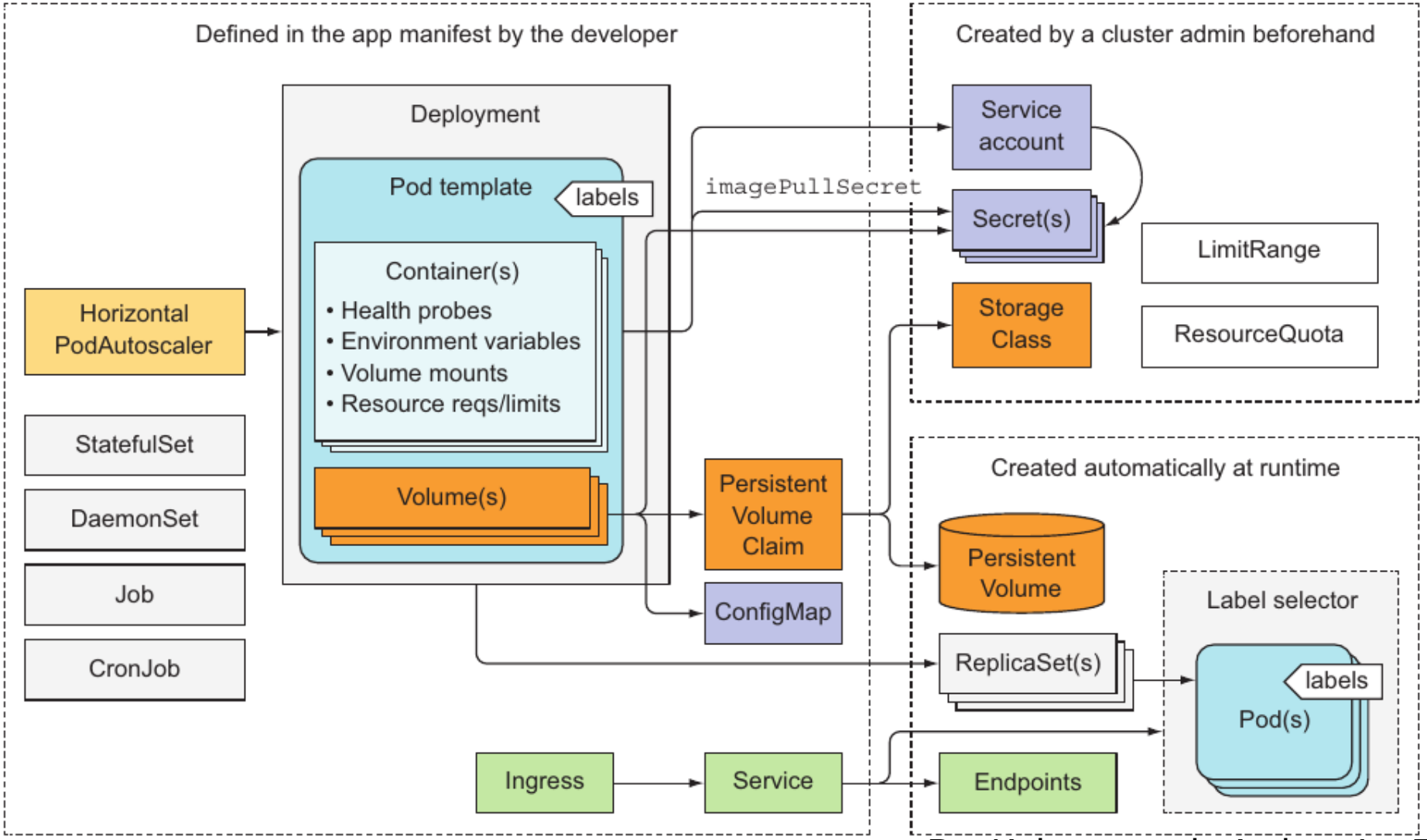
# Next Step: Helm Charts



# Snippet: Service Definition

```yaml
apiVersion: v1
kind: Service
metadata:
  name: {{ include "spring-boot-demo.fullname" . }}
  namespace: {{ include "spring-boot-demo.namespaceName" . }}
  labels:
    {{- include "spring-boot-demo.labels" . | nindent 4 }}
spec:
  type: {{ .Values.service.type }}
  ports:
    - port: {{ .Values.service.port }}
      targetPort: 8080
      protocol: TCP
      name: http
  selector:
    {{- include "spring-boot-demo.selectorLabels" . | nindent 4 }}
```

# Which K8s resource should I care about as a dev?



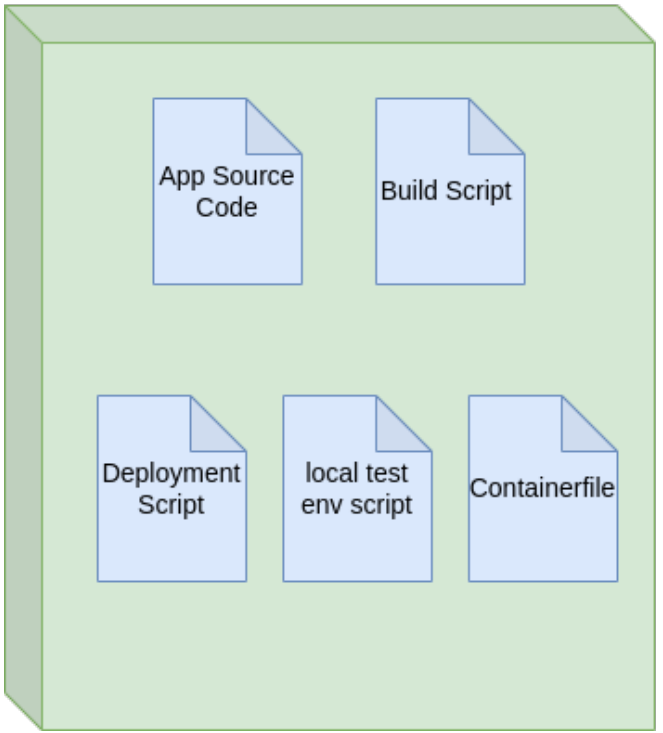

By: Kubernetes in Action, 1st Edition

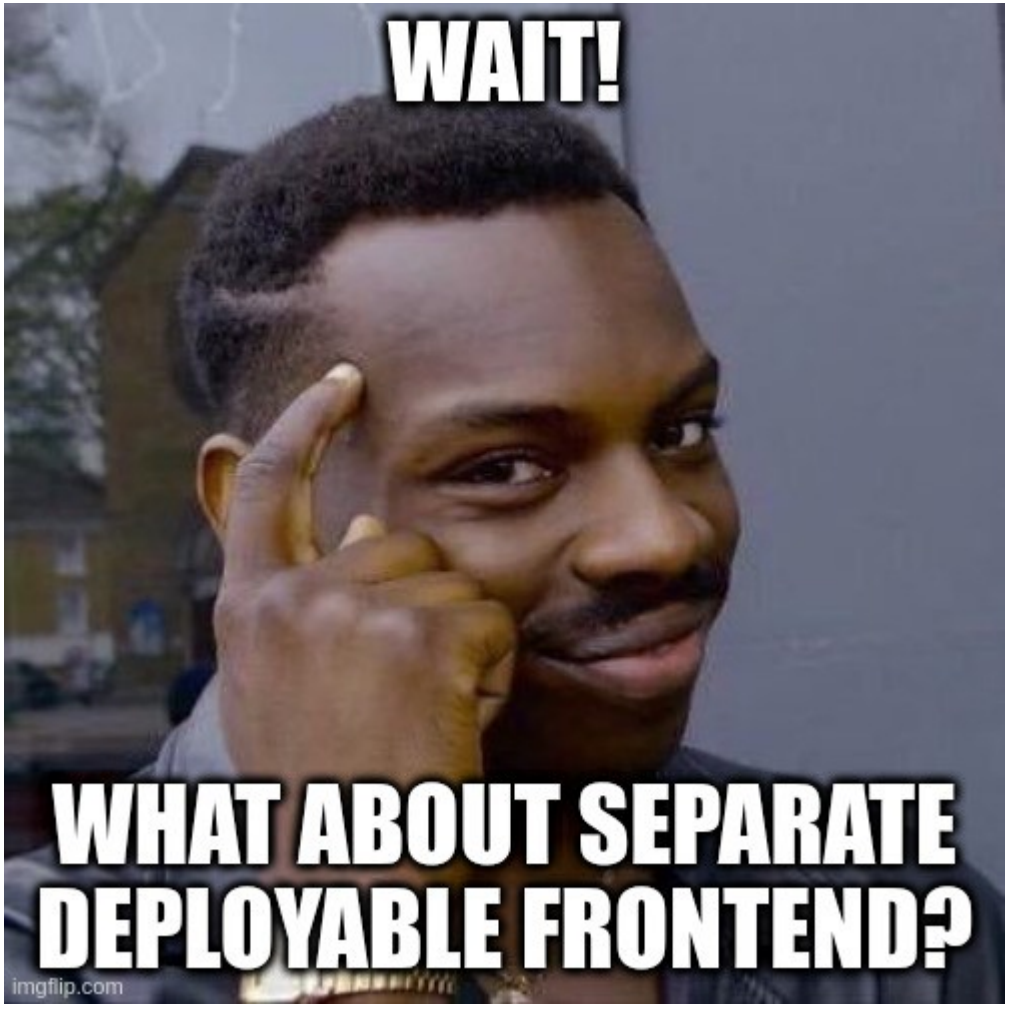
```xml
<plugin>
    <groupId>io.kokuwa.maven</groupId>
    <artifactId>helm-maven-plugin</artifactId>
    <version>6.3.0</version>
    <configuration>
        <chartDirectory>${project.basedir}/deployment/helm-charts</chartDirectory>
        <chartVersion>${project.version}</chartVersion>
        <helmVersion>3.8.1</helmVersion>
    </configuration>
    <executions>
        <execution>
            <id>build-chart</id>
            <phase>package</phase>
            <goals>
                <goal>package</goal>
            </goals>
        </execution>
        <execution>
            <id>upload-chart</id>
            <phase>deploy</phase>
            <goals>
                <goal>upload</goal>
            </goals>
      </execution>
    </executions>
</plugin>
```

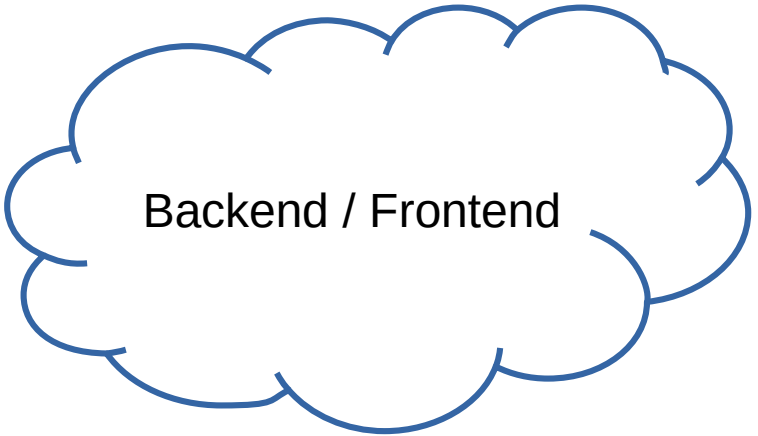
Helm Charts packaging is also
a part of the build process

# Helm Chart Repository

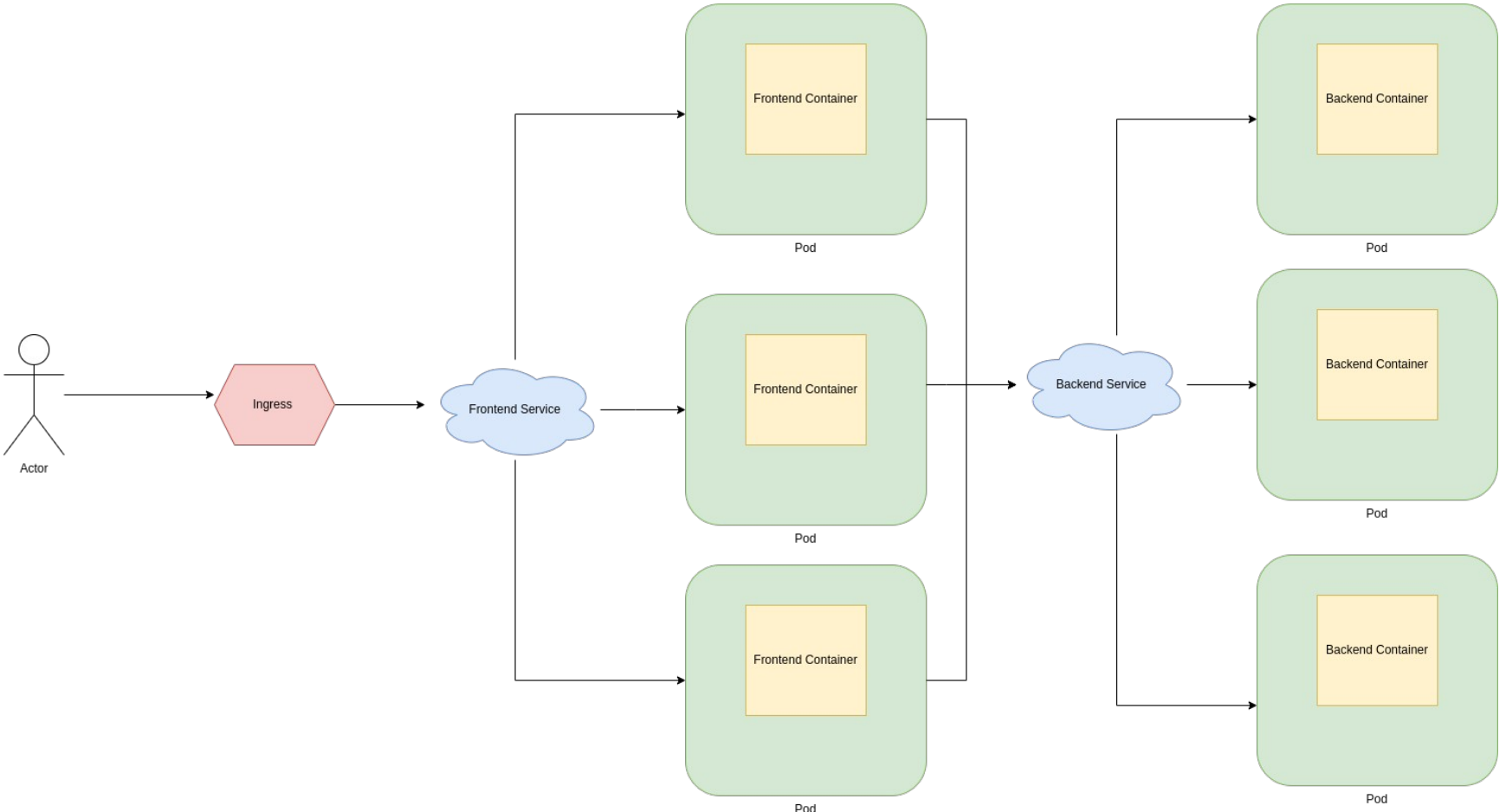
- In Common:
  - Any container registry can be used for this purpose
- Specialized in this:
  - Chartmuseum
  - JFrog Container Registry
  - Artifactory
  - Sonatype Nexus


Application Git Repository
App Source Code
Build Script
Deployment Script
local test env script
Containerfile

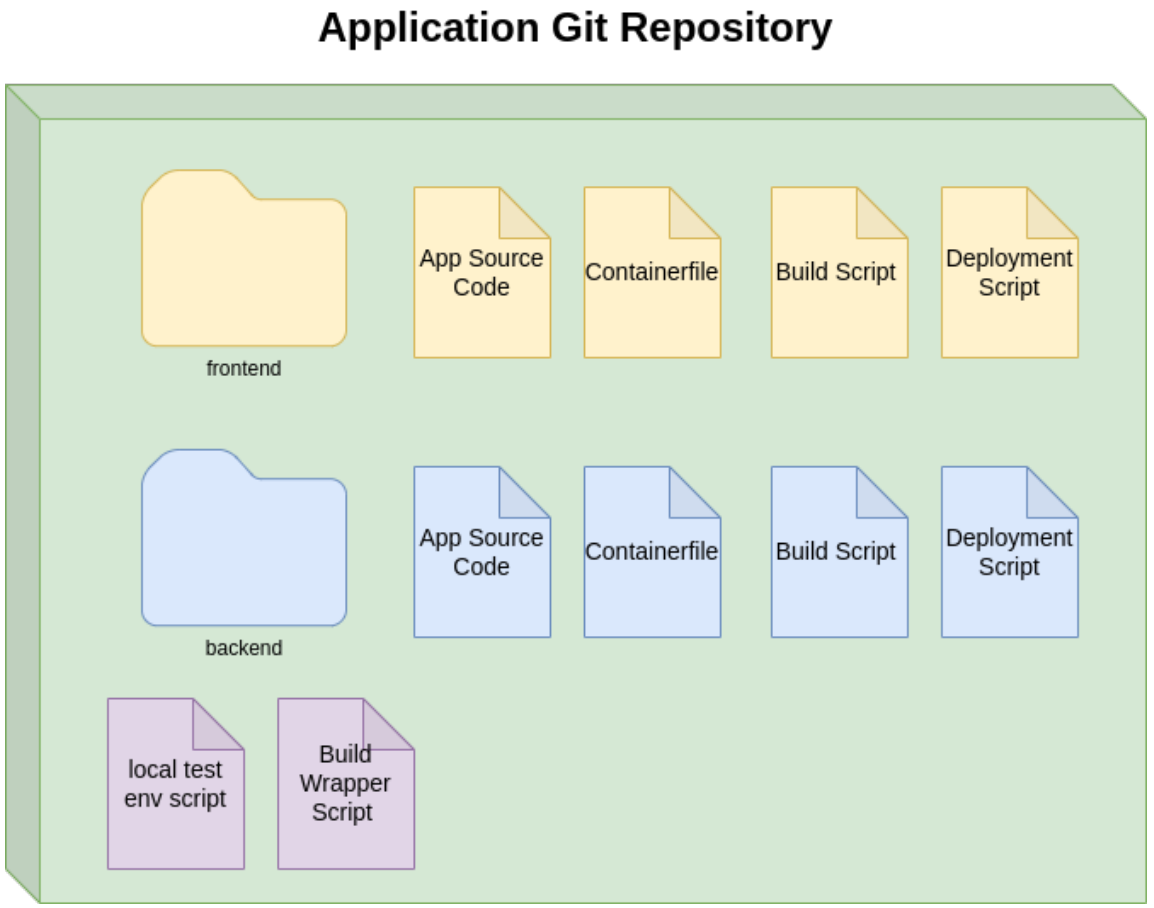

WAIT!
WHAT ABOUT SEPARATE DEPLOYABLE FRONTEND?
imgflip.com


Backend / Frontend

# Frontend And Backend in K8s



# Git Repository Structure



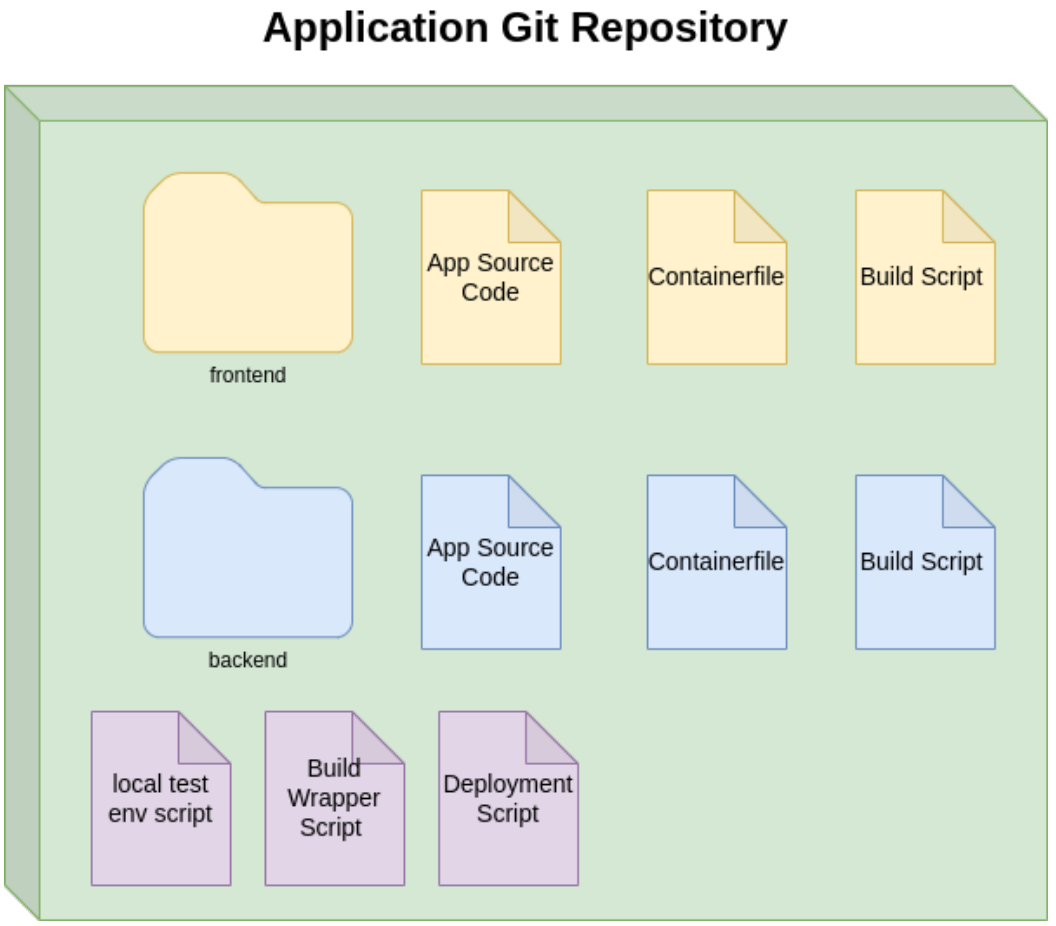

Deployment Script

- Ingress
- Frontend Service
- Frontend Deployment

Deployment Script

- Backend Service
- Backend Deployment

# Git Repository Structure



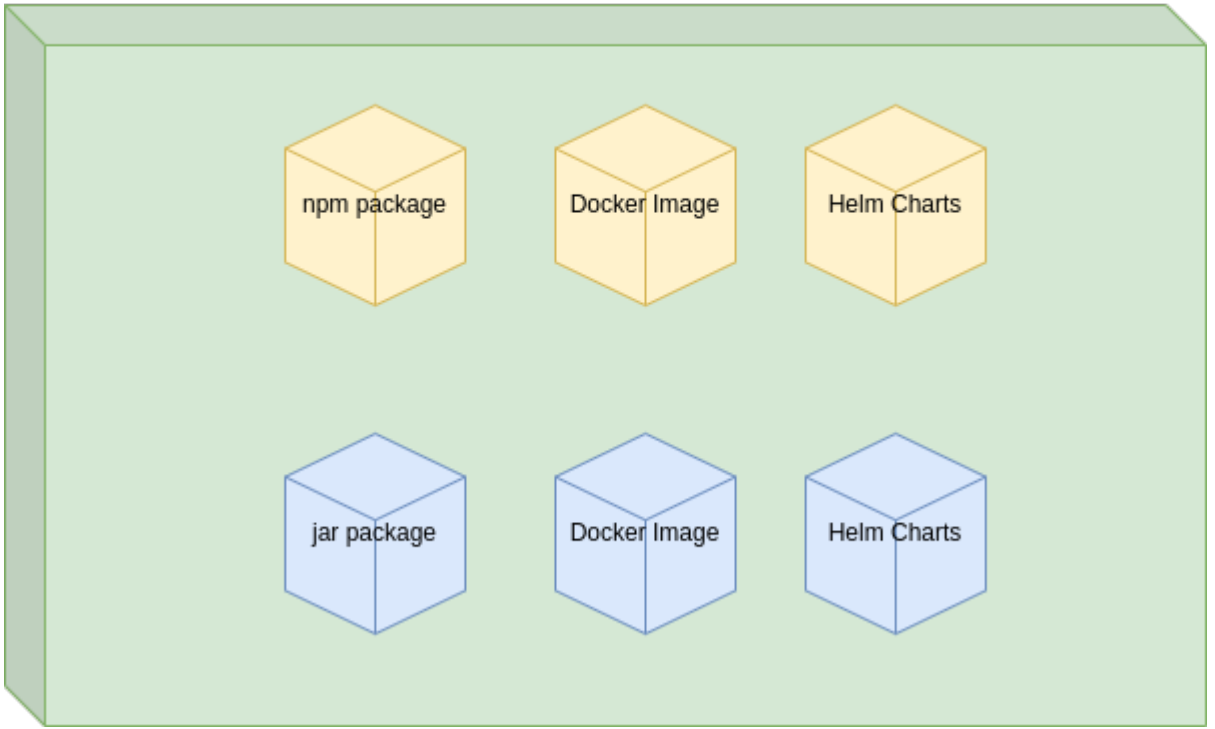

Deployment Script

- Ingress
- Frontend Service
- Frontend Deployment
- Backend Service
- Backend Deployment

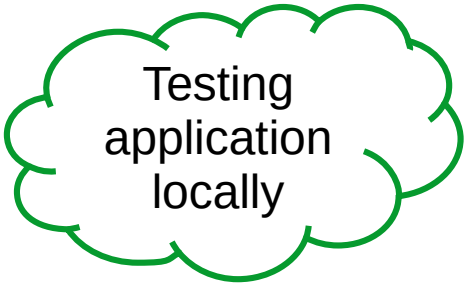

Versioning

# Start Simple:
# One version number for all artifacts

# Application Artifacts

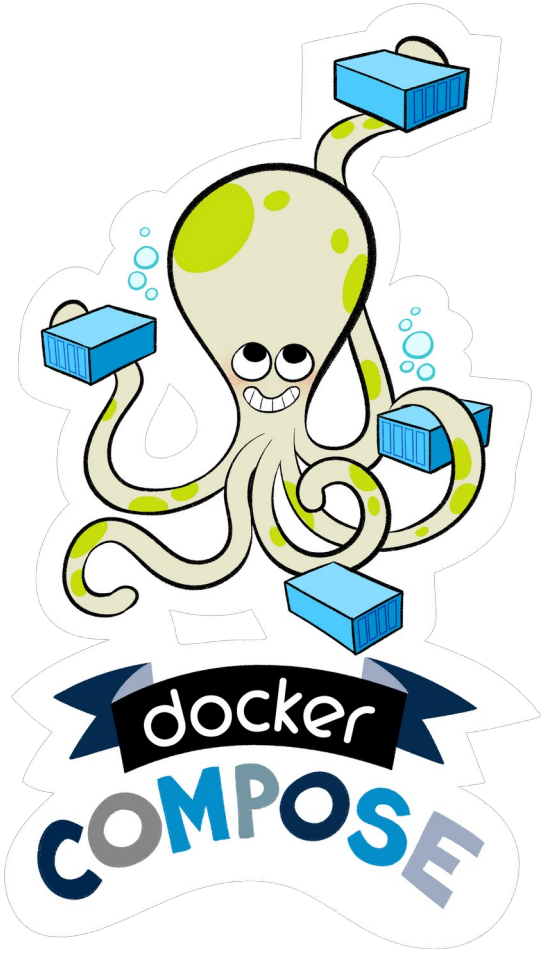



Locale Development
Environment


Testing
application
locally


Developing
deployment
scripts locally

# Testing Application Locally





Spring Boot Maven Plugin

# Testing Application Locally

```yaml
version: "3.9"
services:
  database:
    image: mongo:4.2.21
    restart: always
    ports:
      - 27017:27017
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: root123
    volumes:
      - ./local-env/:/dockerentrypoint-initdb.d/
```

```xml
        <plugin>
               <groupId>org.springframework.boot</groupId>
               <artifactId>spring-boot-maven-plugin</artifactId>
               <configuration>
                   <layers>
                       <enabled>true</enabled>
                   </layers>
                   <environmentVariables>
                       <MONGODB_ENABLED>true</MONGODB_ENABLED>
                       <MONGODB_URI>mongodb://test:test123@localhost/test</MONGODB_URI>
                   </environmentVariables>
               </configuration>
        </plugin>
```

```yaml
version: "3.9"
services:
  demo-app:
    image: spring-boot-demo:latest
    restart: always
    ports:
      - 80:8080
    environment:
      MONGODB_ENABLED: "true"
      MONGODB_URI: mongodb://test:test123@database:27017/test
    depends_on:
      - database

  database:
    image: mongo:4.2.21
    restart: always
    ports:
      - 27017:27017
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: root123
    volumes:
      - ./local-env/:/docker-entrypoint-initdb.d/
```


Other
dependencies?

# Mocking

- https://www.mock-server.com
- https://github.com/navikt/mock-oauth2-server

```yaml
version: "3.9"
services:
  mockserver:
    image: mockserver/mockserver:latest
    restart: always
    ports:
      - 1080:1080
    environment:
      MOCKSERVER_INITIALIZATION_JSON_PATH: /config/expectation.json
    volumes:
      - ./local-env/mockserver:/config
```

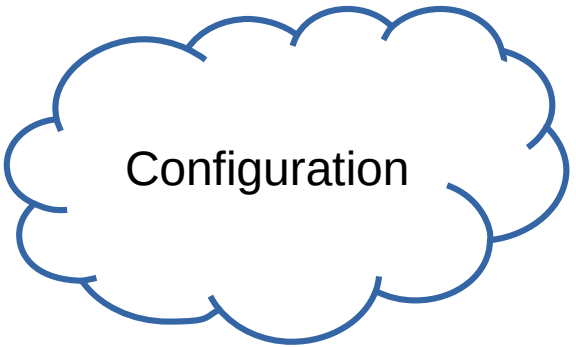
```
[
  {
    "httpRequest": {
      "path": "/success"
    },
    "httpResponse": {
      "body": "Successful!"
    }
  },
  {
    "httpRequest": {
      "path": "/fail"
    },
    "httpResponse": {
      "statusCode": 400
    }
  }
]
```

# Developing deployment scripts locally

minikube

# Alternatives for Minikube

- k3s
- k3d
- kind
- microk8s
- k0s


Configuration

# 12 Factor App:
# Store the configuration in environment variables

# Prepare the application


```
# snippet application.properties

spring.data.mongodb.uri=${MONGODB_URI:mongodb://localhost/test}

mongodb.enabled=${MONGODB_ENABLED:false}
```

# Adjust Helm Charts

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ include "spring-boot-demo.fullname" . }}-config
  namespace: {{ include "spring-boot-demo.namespaceName" . }}
  labels:
  {{- include "spring-boot-demo.labels" . | nindent 4 }}
data:
  MONGODB_URI: "{{ .Values.mongodb.uri }}"
  MONGODB_ENABLED:  "{{ .Values.mongodb.enabled }}"
```

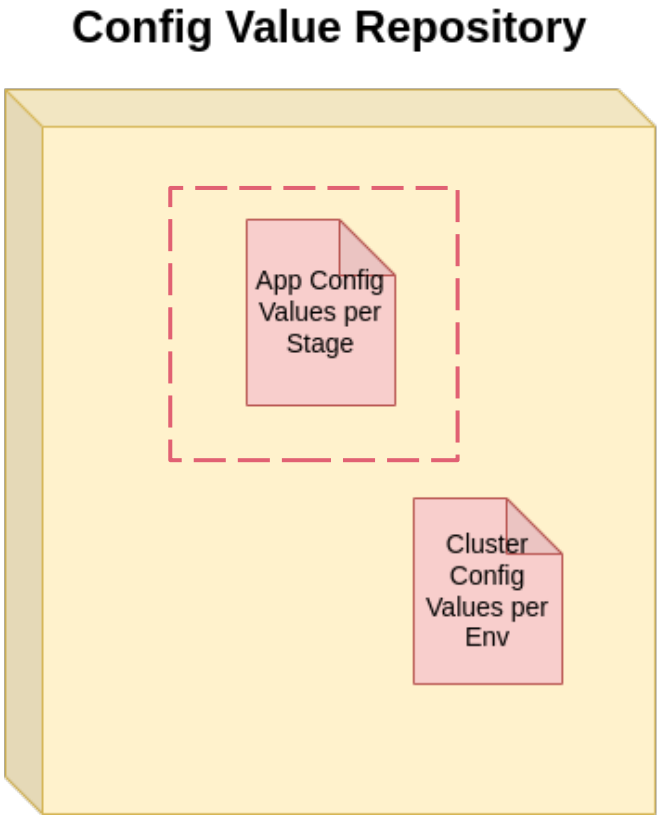
# Adjust Helm Charts

```
# code snippet with the important part
apiVersion: apps/v1
kind: Deployment
# ...
spec:
  template:
    metadata:
      annotations:
        checksum/config: {{ include (print $.Template.BasePath "/config.yaml") . | sha256sum }}
    spec:
      containers:
      - name: {{ .Chart.Name }}
        image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
        args: [{{ .Values.spring_boot_demo_chart.container_args }}]
        imagePullPolicy: {{ .Values.image.pullPolicy }}
        envFrom:
          - configMapRef:
              name: {{ include "spring-boot-demo.fullname" . }}-config
```

# Adjust Helm Charts

```yaml
# code snippet with the important part from value.yaml
mongodb:
  enabled: false
  uri: mongodb://test:test@localhost/test
```

# Manage Configuration

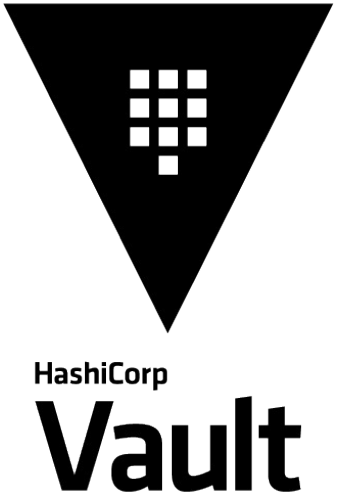


# Manage Configuration

```
config-value-repo on  dev
➜ tree
.
├── namespace-a
│   └── app1.yml
└── registry.yaml

➜ git branch
* dev
  pre-prod
  prod
```

```
flat-config-value-repo on  master
➜ tree
.
├── dev
│   ├── namespace-a
│   │   └── app1.yml
│   └── registry.yaml
├── pre-prod
│   ├── namespace-a
│   │   └── app1.yml
│   └── registry.yaml
└── prod
    ├── namespace-a
    │   └── app1.yml
    └── registry.yaml
```

# Secrets




Cloud Solutions (Ex.):

- Google Secret Manager
- AWS Secrets & Configuration Provider
- Azure Key Vault Provider

# Helm Secret Plugin

```
→ helm plugin install https://github.com/jkroepke/helm-secrets --version v3.12.0
→ helm secrets help

Secrets encryption in Helm Charts

This plugin provides ability to encrypt/decrypt secrets files
to store in less secure places, before they are installed using
Helm.

For more information, see the README at github.com/jkroepke/helm-secrets

To decrypt/encrypt/edit you need to initialize/first encrypt secrets with
sops - https://github.com/mozilla/sops
```

# Helm Secret Plugin

```
// sops must be configured
➜ helm secrets enc examples/sops/secrets.yaml
Encrypting examples/sops/secrets.yaml
Encrypted examples/sops/secrets.yaml
➜ helm upgrade name . -f secrets://examples/sops/secrets.yaml value.yaml
```


Make app K8s-ready

# Good Practices for Applications in Container

- Only one application process per container
- Avoid execution as root
- Avoid privileged containers
- Prefer stateless applications
- Logging messages on stdout
- Consider application monitoring
- Be able to start and stop robustly

# Logging messages on stdout


Hint: Use Spring Default Logging Settings

# Application Monitoring


```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

```
management.metrics.export.prometheus.enabled=true
management.metrics.web.server.request.autotime.enabled=true
management.endpoints.web.exposure.include=prometheus
```

# Start and Stop Robustly


```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

```
management.endpoints.web.exposure.include=info,health
```

# Start and Stop Robustly

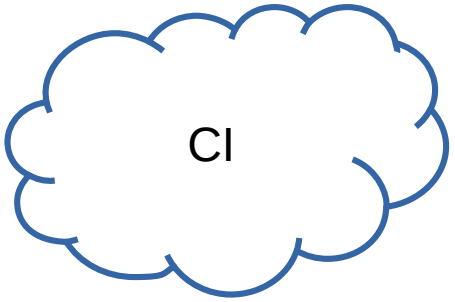
```yaml
# code snippet with the important part
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      containers:
      - name: {{ .Chart.Name }}
        image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
        ports:
        - name: container-http
          containerPort: 8080
          protocol: TCP
        livenessProbe:
          httpGet:
            path: /actuator/health/liveness
            port: container-http
          initialDelaySeconds: {{ .Values.livenessProbe.initialDelaySeconds }}
          periodSeconds: {{ .Values.livenessProbe.periodSeconds }}
          timeoutSeconds: {{ .Values.livenessProbe.timeoutSeconds }}
        readinessProbe:
          httpGet:
            path: /actuator/health/readiness
            port: container-http
          initialDelaySeconds: {{ .Values.readinessProbe.initialDelaySeconds }}
          periodSeconds: {{ .Values.readinessProbe.periodSeconds }}
          timeoutSeconds: {{ .Values.readinessProbe.timeoutSeconds }}
```

# Start and Stop Robustly

Important:
Secure these endpoints to the outside!

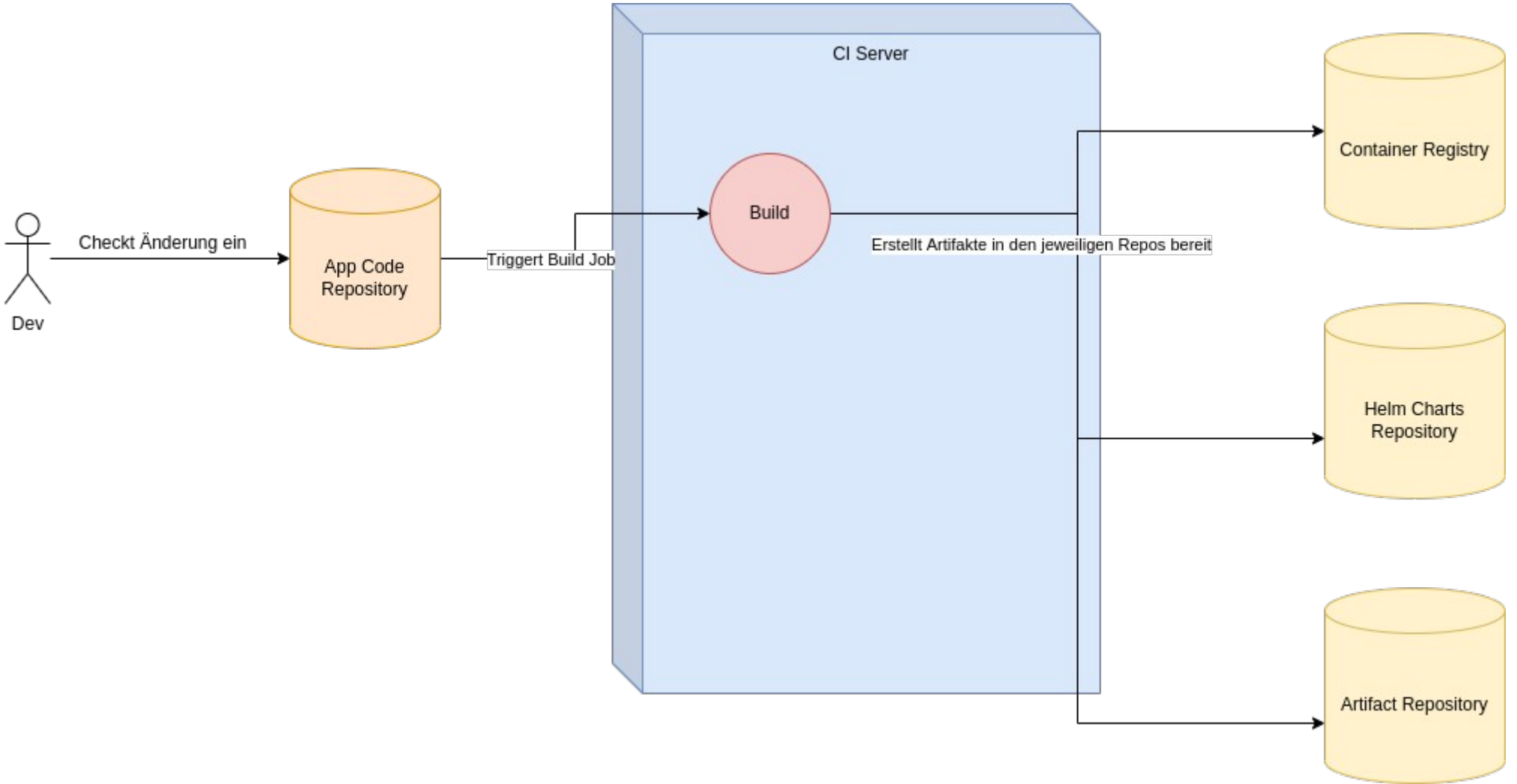
# Start and Stop Robustly

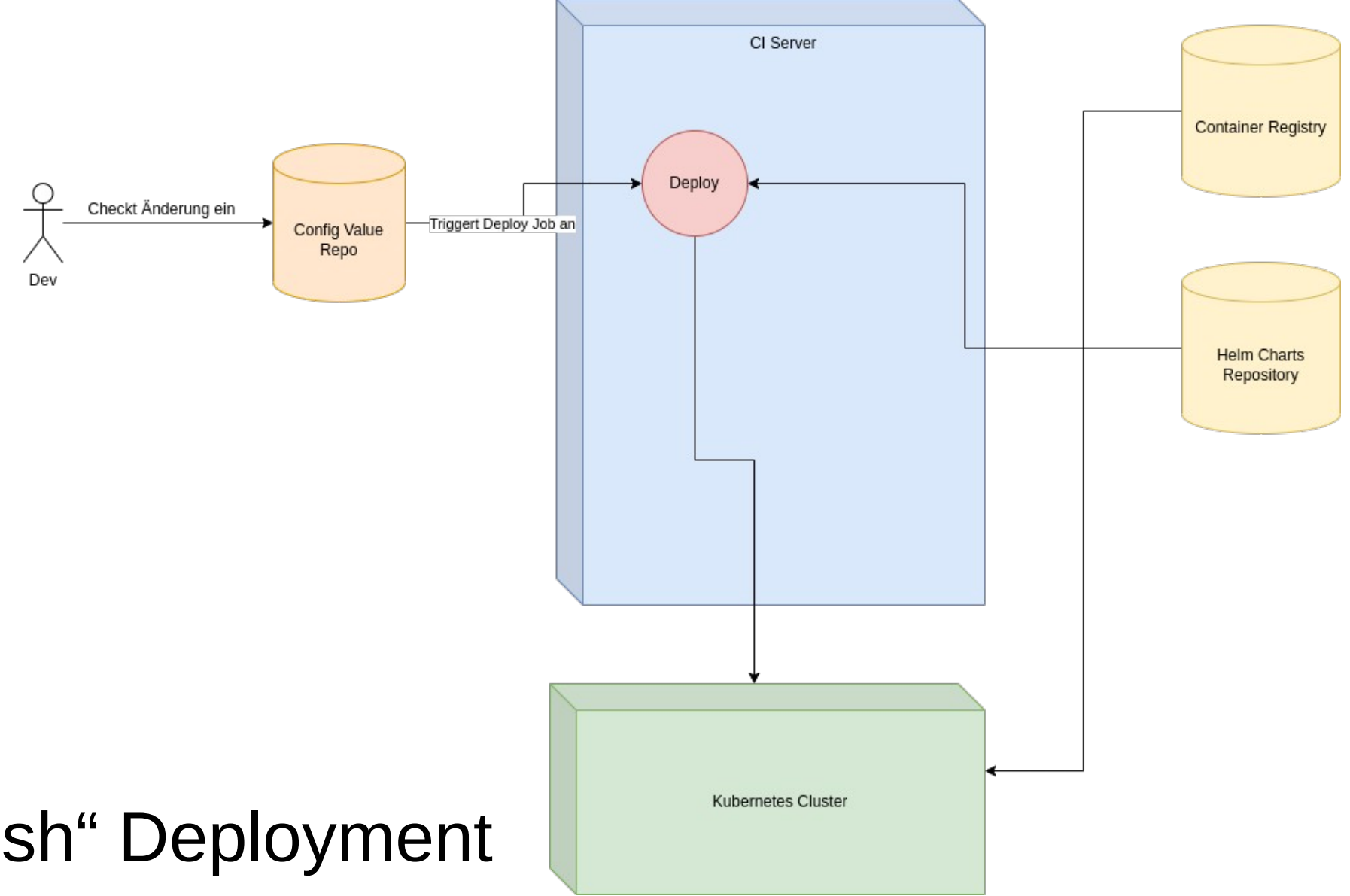
```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: {{ include "spring-boot-demo.fullname" . }}
  namespace: {{ include "spring-boot-demo.namespaceName" . }}
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$1
    nginx.ingress.kubernetes.io/x-forwarded-prefix: "/"
    nginx.ingress.kubernetes.io/server-snippet: |
      location ~* "^/actuator/" {
          deny all;
          return 403;
        }
spec:
  rules:
    - host: {{ .Values.ingress.host }}
      http:
        paths:
          - path: /(.*)
            pathType: Prefix
            backend:
              service:
                name: {{ include "spring-boot-demo.fullname" . }}
                port:
                  number: 8080
```


Cl

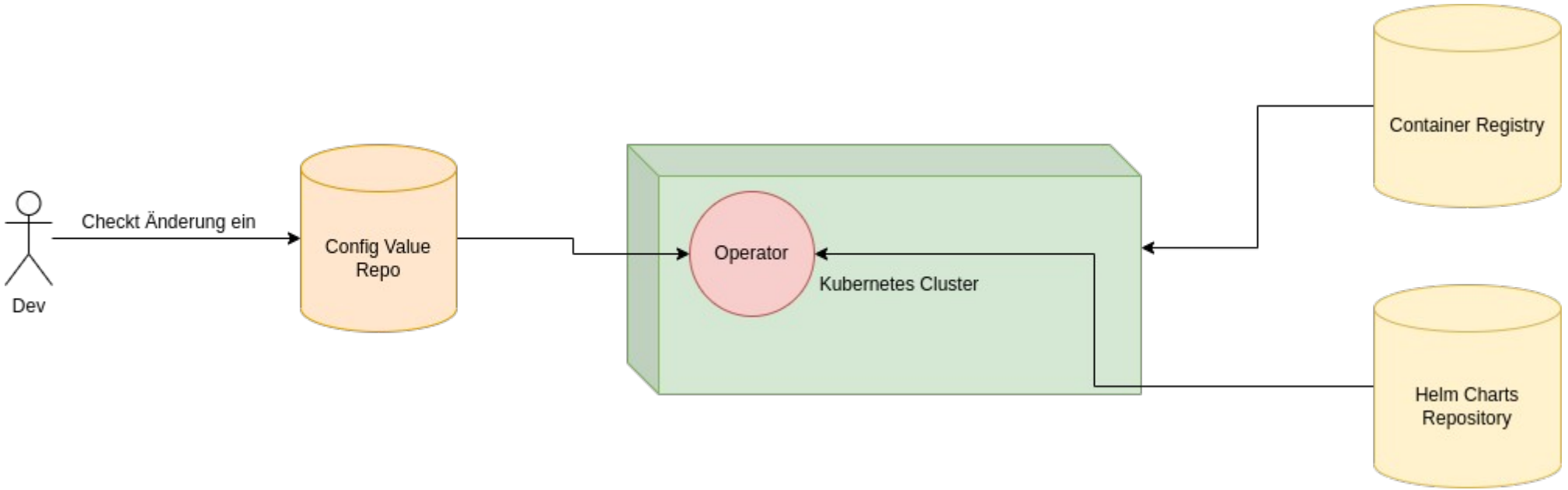
# 12 Factor App:
# Strictly separate build and run phase

# Build

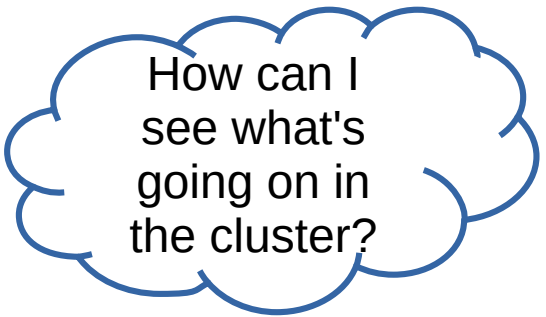

CI Server
Container Registry
Build
Checkt Änderung ein
App Code Repository
Triggert Build Job
Erstellt Artifakte in den jeweiligen Repos bereit
Dev
Helm Charts Repository
Artifact Repository




# „Push“ Deployment



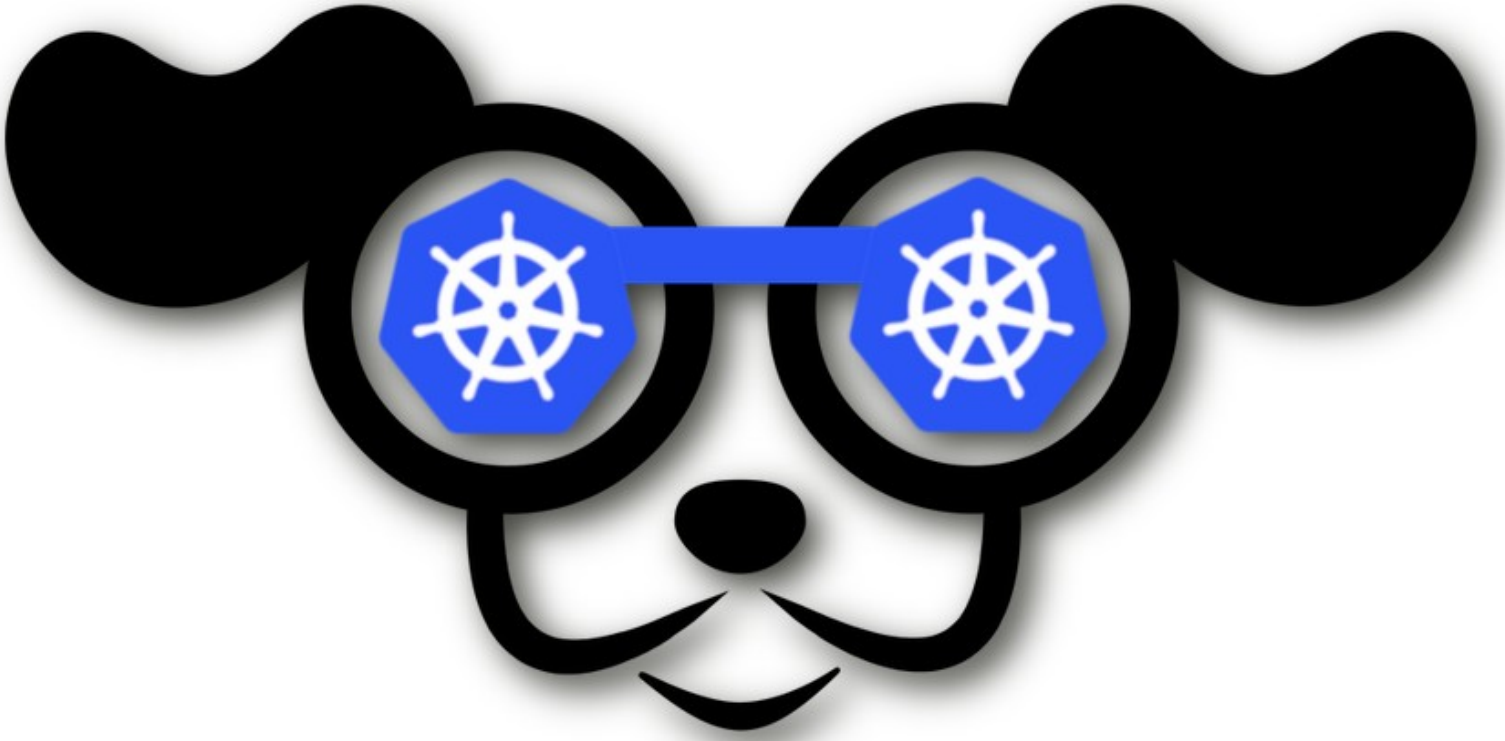

# „Pull“ Deployment


How can I see what's going on in the cluster?

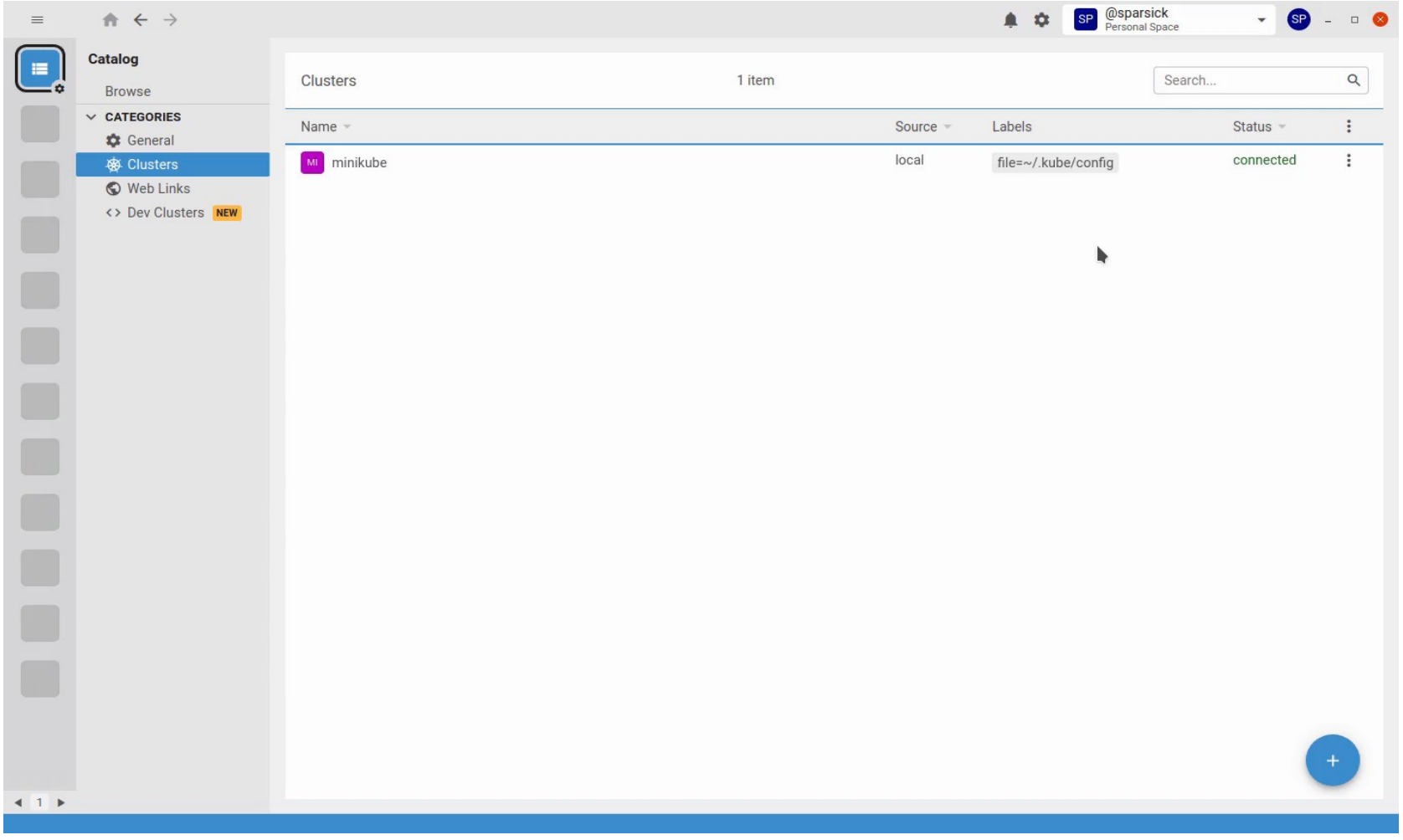
# kubectl

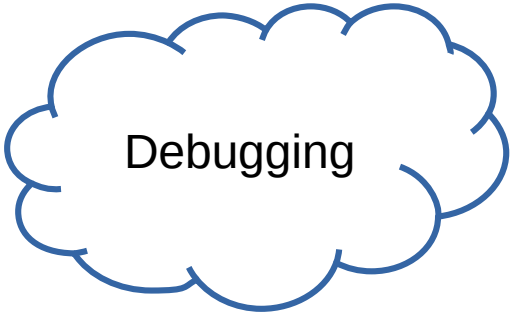
# k9s

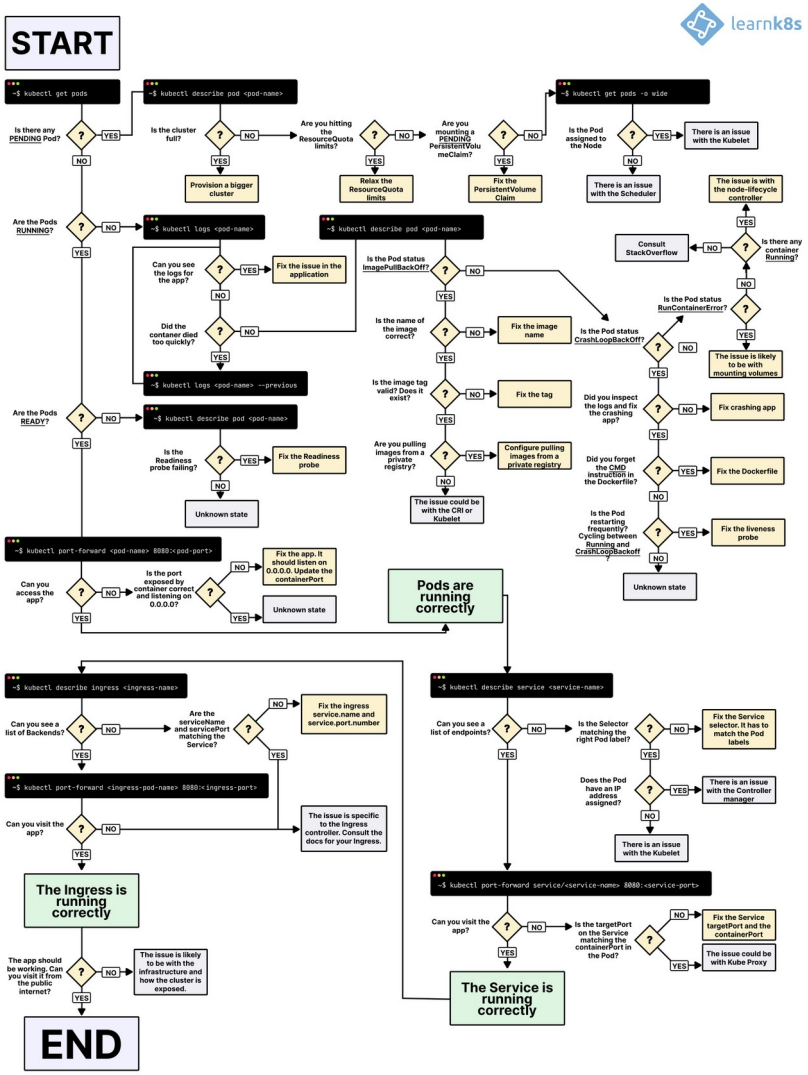
*Kubernetes CLI To Manage Your Clusters In Style!*



# K8s Lens


@sparsick
Personal Space
Catalog
Browse
CATEGORIES
General
Clusters
Web Links
Dev Clusters
NEW
Clusters
1 item
Search...
Name
Source
Labels
Status
minikube
local
file=~/.kube/config
connected


Debugging




Troubleshooting flowchart https://learnk8s.io/troubleshooting-deployments learnk8s

https://learnk8s.io/troubleshooting-deployments

# Troubleshooting Applications

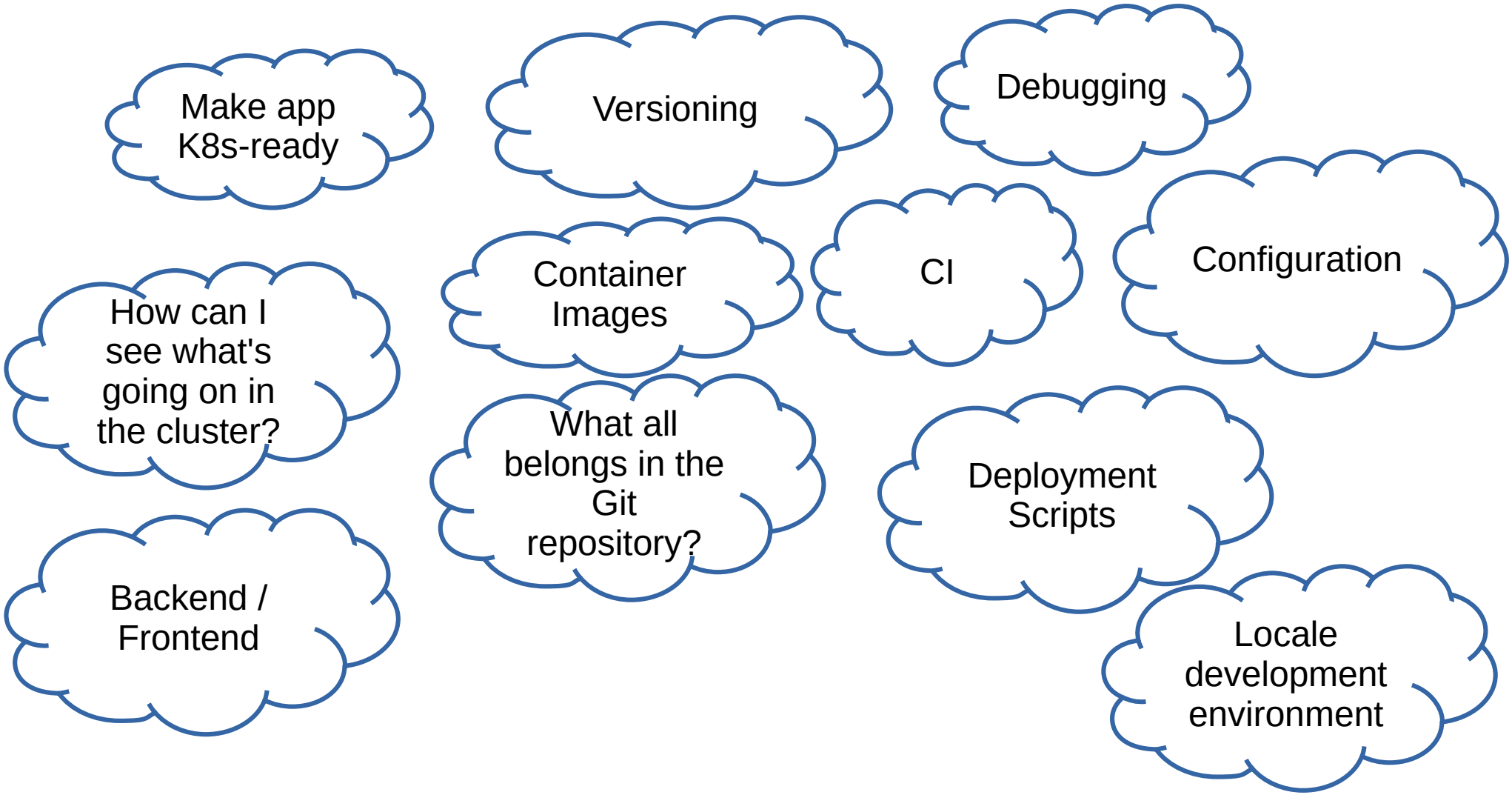
This doc contains a set of resources for fixing issues with containerized applications. It covers things like common issues with Kubernetes resources (like Pods, Services, or StatefulSets), advice on making sense of container termination messages, and ways to debug running containers.



https://kubernetes.io/docs/tasks/debug/debug-application/

# debug container (K8s v1.23)

```
$ kubectl run ephemeral-demo --image=k8s.gcr.io/pause:3.1 --restart=Never

$ kubectl exec -it ephemeral-demo -- sh
OCI runtime exec failed: exec failed: container_linux.go:346: starting container process caused "exec:
\"sh\": executable file not found in $PATH": unknown

$ kubectl debug -it ephemeral-demo --image=busybox:1.28 --target=ephemeral-demo
Defaulting debug container name to debugger-8xzrl.
If you don't see a command prompt, try pressing enter.
/ #
```

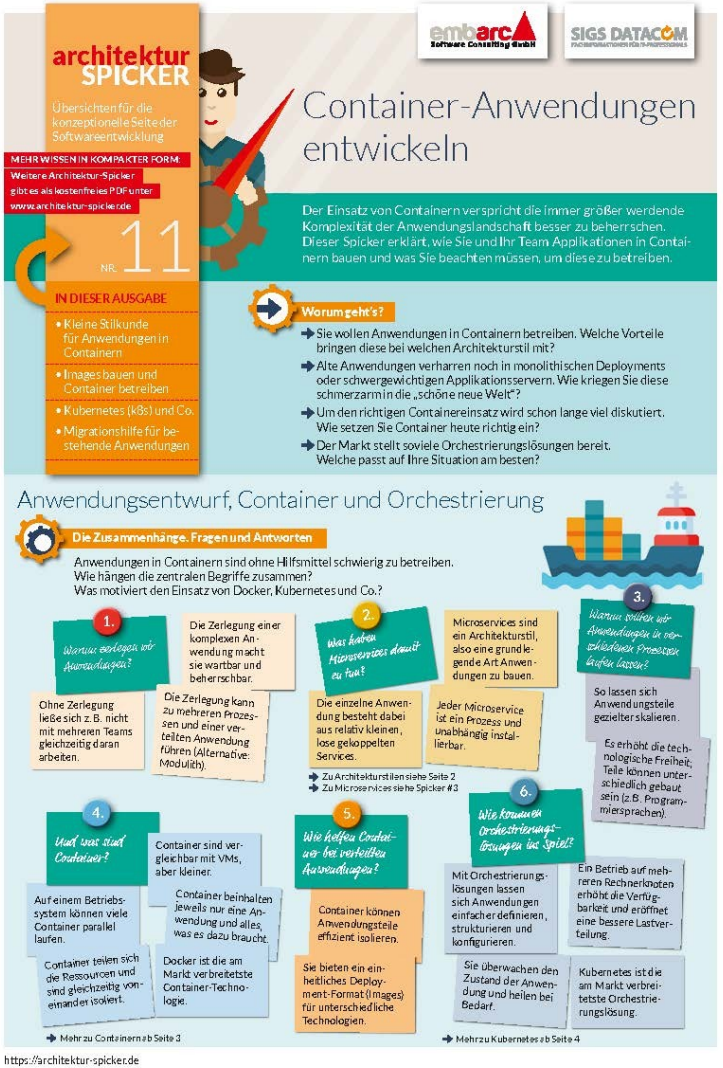

Make app K8s-ready
Versioning
Debugging
Container Images
CI
Configuration
How can I see what's going on in the cluster?
What all belongs in the Git repository?
Deployment Scripts
Backend / Frontend
Locale development environment

Questions?

mail@sandra-parsick.de

@SandraParsick

https://github.com/sparsick/k8s-dev-survival-kit-talk




https://architektur-spicker.de

# Further Informations

- https://www.informatik-aktuell.de/entwicklung/methoden/container-images-deep-dive-101-wege-zum-bauen-und-bereitstellen.html
- „Kubernetes in Action“ by Marko Lukša
- „Docker in Action“ by Jeff Nickoloff, Stephen Kuenzli
- „Container-Anwendungen entwickeln“ https://www.architektur-spicker.de/ (in German)
- „Continuous Delivery“ https://www.architektur-spicker.de/ (in German)

# Pictures Credits

- https://unsplash.com/photos/RfwGg5ZZh4Q?utm_source=unsplash&utm_medium=referral&utm_content=creditShareLink
- https://unsplash.com/photos/CpsTAUPoScw?utm_source=unsplash&utm_medium=referral&utm_content=creditShareLink