ATLASSIAN

team'22

Find vulnerabilities before security knocks on your door

Wearing belts and suspenders

Marco Morales, Partner Solutions Architect, Snyk
Eric Smalling, Sr. Developer Advocate, Snyk

Today’s Plan

Set the goals for today

Setting context

Common problems and misconceptions

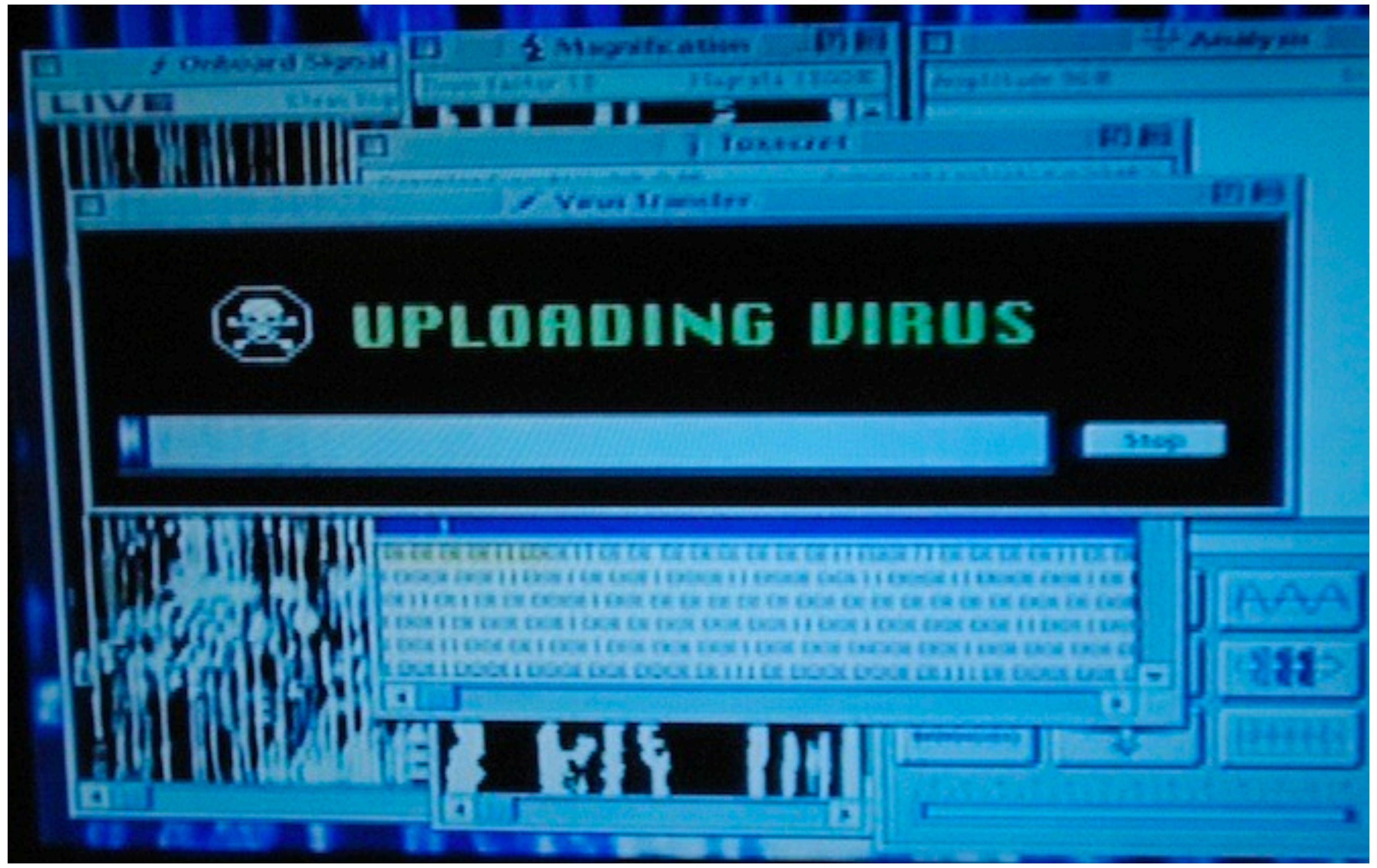
Use cases and scenarios

Conclusions and wrap-up

TODAY’S GOAL

We’ll take a journey with real examples to show security doesn’t have to be scary

Context



Consider Log4j and Log4Shell

Scary because attackers could run *anything on your system*

In the movies

Launch missiles, release contagions, destroy alien spaceships

In the real world

Data breaches, loss of data, unwanted applications

Common problems and misconceptions

Security is hard

Too much time

Too much work

Too many tools and tasks

Tool confusion

I already have a firewall

One tool to rule them all

False outcomes. Prioritization. Red herrings.

People don’t care

People do care – A LOT

Use cases

Examine

Exploit

Fix

Vulnerable Application

Custom Code, Open Source, Habits

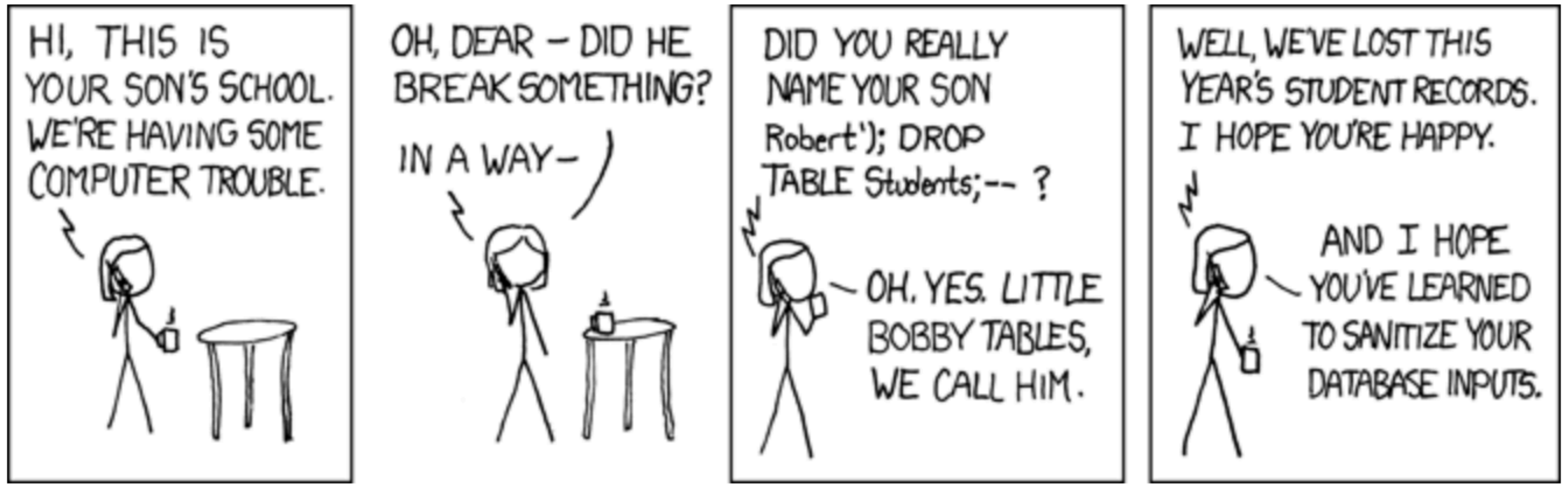
Vulnerable Container

Image-level settings

Suboptimal Infrastructure

Misconfigurations

Vulnerable Applications


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Source: https://xkcd.com/327/

Source Code

Your teams write great code.

They may introduce vulnerabilities.

Open Source

80-90% of a modern application is open-source

You don’t always know what you bring in

Habits

Add software security to your daily activities and CI/CD pipeline

Vulnerable Applications

Source Code

Open Source

Habits

The OWASP Foundation provides great resources

- Examples and mitigation guidance
- OWASP Top-10

https://owasp.org/www-project-top-ten/

Consider a SQL Injection

```
String query =
"SELECT \* FROM accounts WHERE custID=`"
+ request.getParameter("id")
+ “'";
```

Attack your software (or scan, or test...)

- If you don’t, somebody else will

Vulnerable Applications

Source Code

Open Source

Habits

Consider the popular Log4j/Log4Shell

- Zero-day vulnerability
- Arbitrary code execution
- Dependency, *also a transitive dependency*
- It seemed like everybody used Log4j

Review how you create and patch your code

- Streamline your build makery and processes
- Iterate (2.15, 2.16, 2.17, …)

How to prepare for the next one?

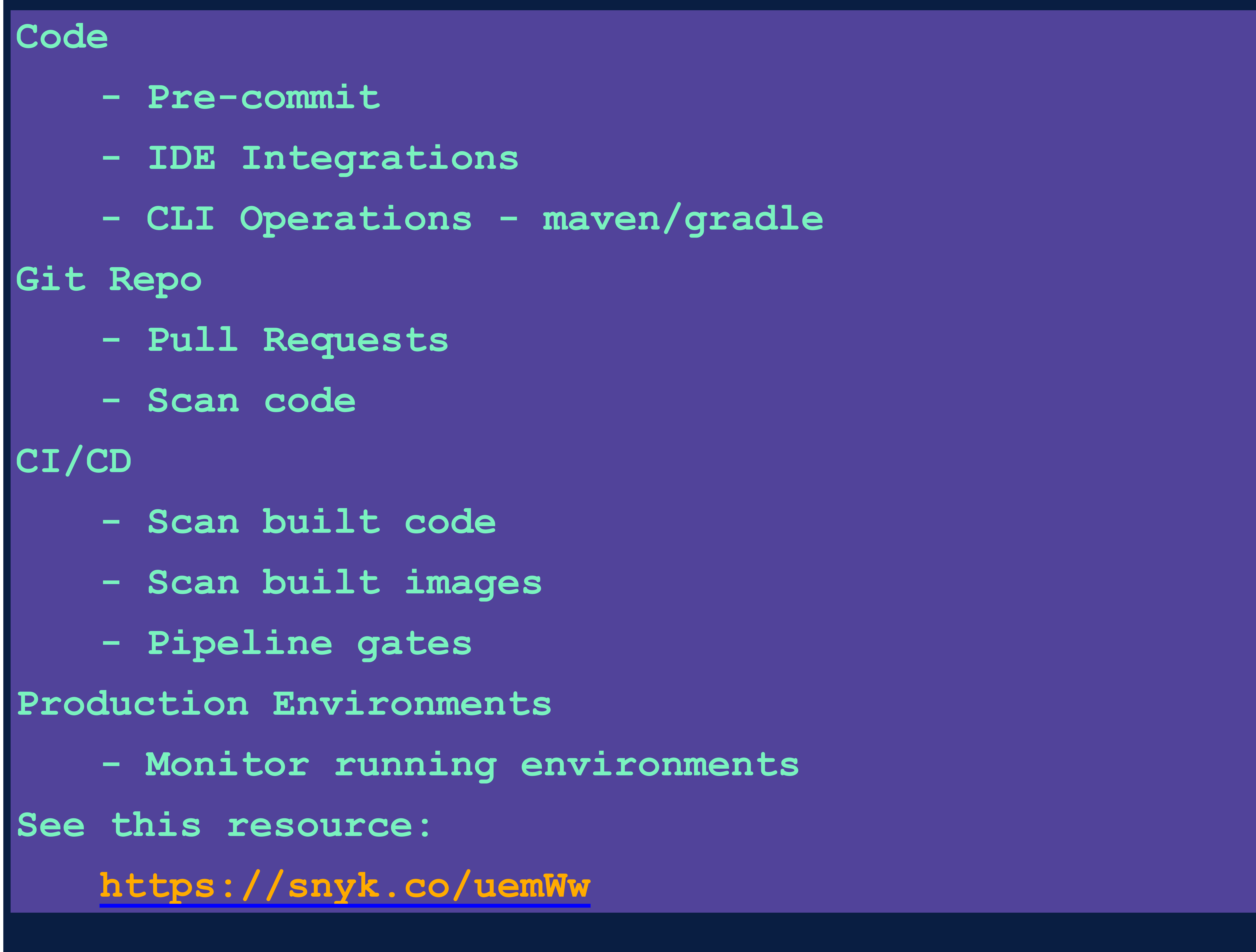
- Be prepared
- Automate processes (build, deploy, test)
- Monitor your repository regularly

Vulnerable Applications

Source Code

Open Source

Habits



Code

- Pre-commit
- IDE Integrations
- CLI Operations - maven/gradle

Git Repo

- Pull Requests
- Scan code

CI/CD

- Scan built code
- Scan built images
- Pipeline gates

Production Environments

- Monitor running environments

See this resource:

https://snyk.co/uemWw

Vulnerable Containers

Image Architecture

What's in your container images matters

Tools & Strategies

Multi-Stage Docker build and other tools

Supply Chain

Know where your images came from and be prepared to prove it

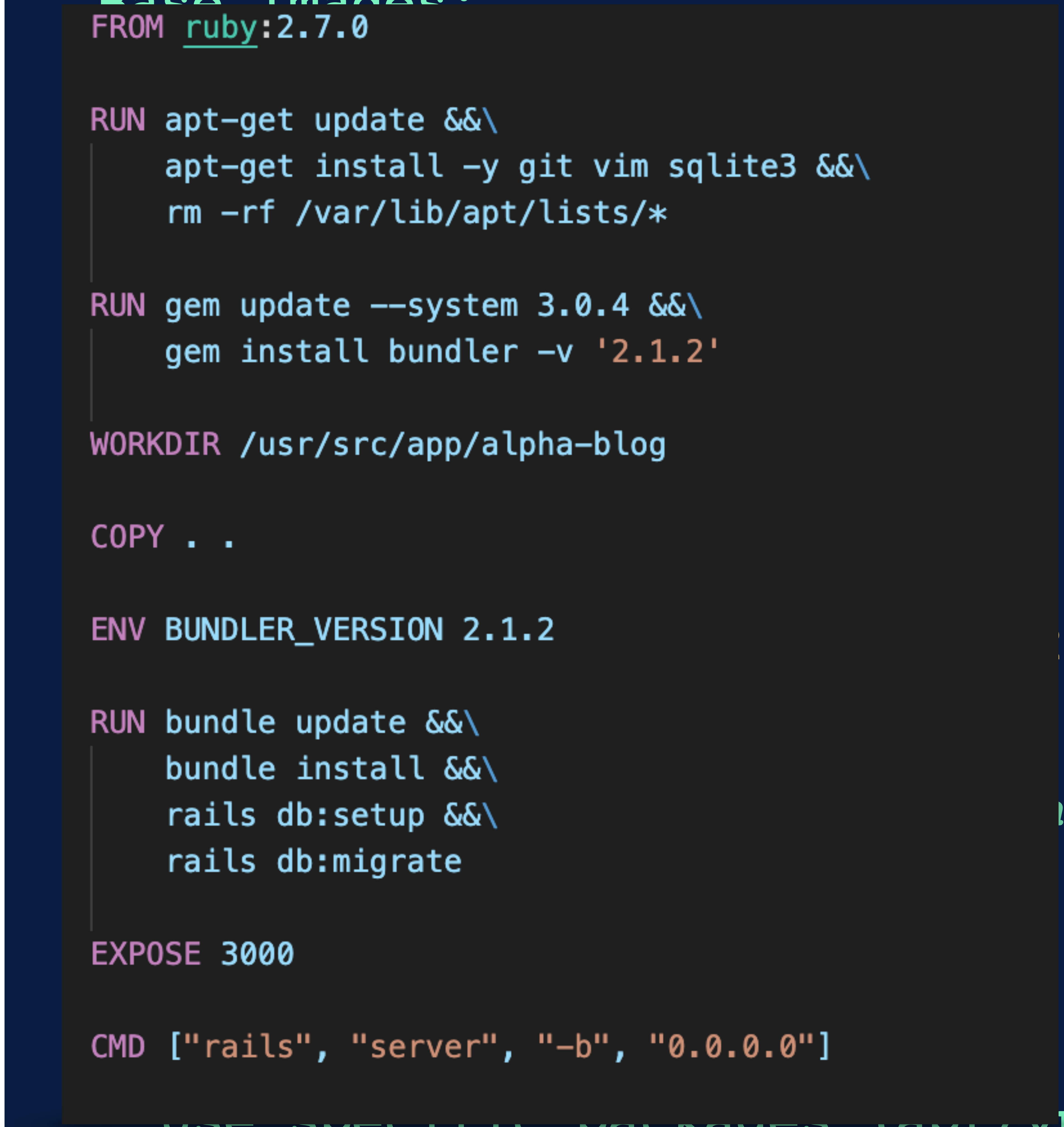
Vulnerable Containers

Image Architecture

Tools & Strategies

Supply Chain

Base images:



```
FROM ruby:2.7.0

RUN apt-get update &&\
    apt-get install -y git vim sqlite3 &&\
    rm -rf /var/lib/apt/lists/*

RUN gem update --system 3.0.4 &&\
    gem install bundler -v '2.1.2'

WORKDIR /usr/src/app/alpha-blog

COPY . .

ENV BUNDLER_VERSION 2.1.2

RUN bundle update &&\
    bundle install &&\
    rails db:setup &&\
    rails db:migrate

EXPOSE 3000

CMD ["rails", "server", "-b", "0.0.0.0"]
```

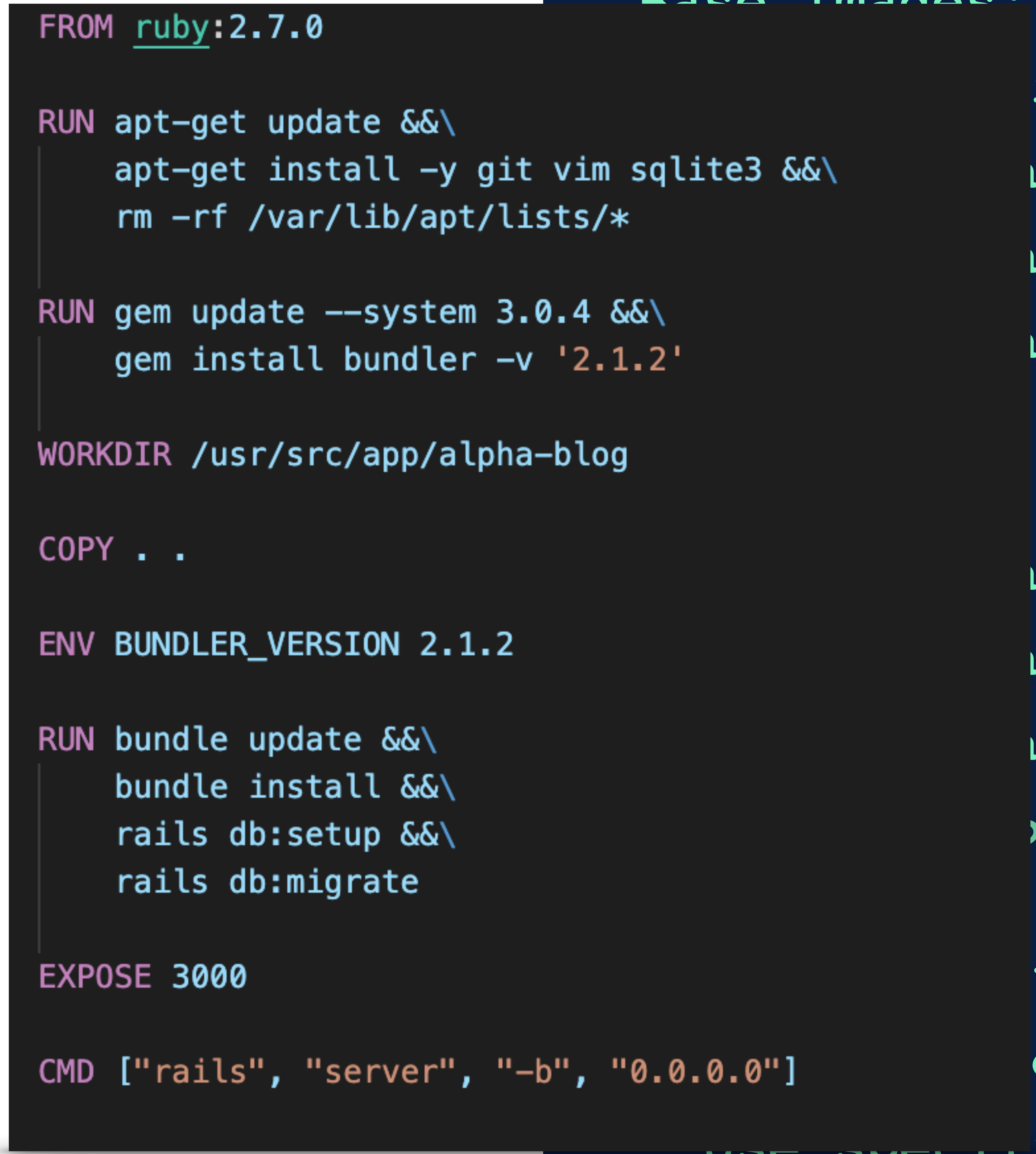
3MB

48.7MB

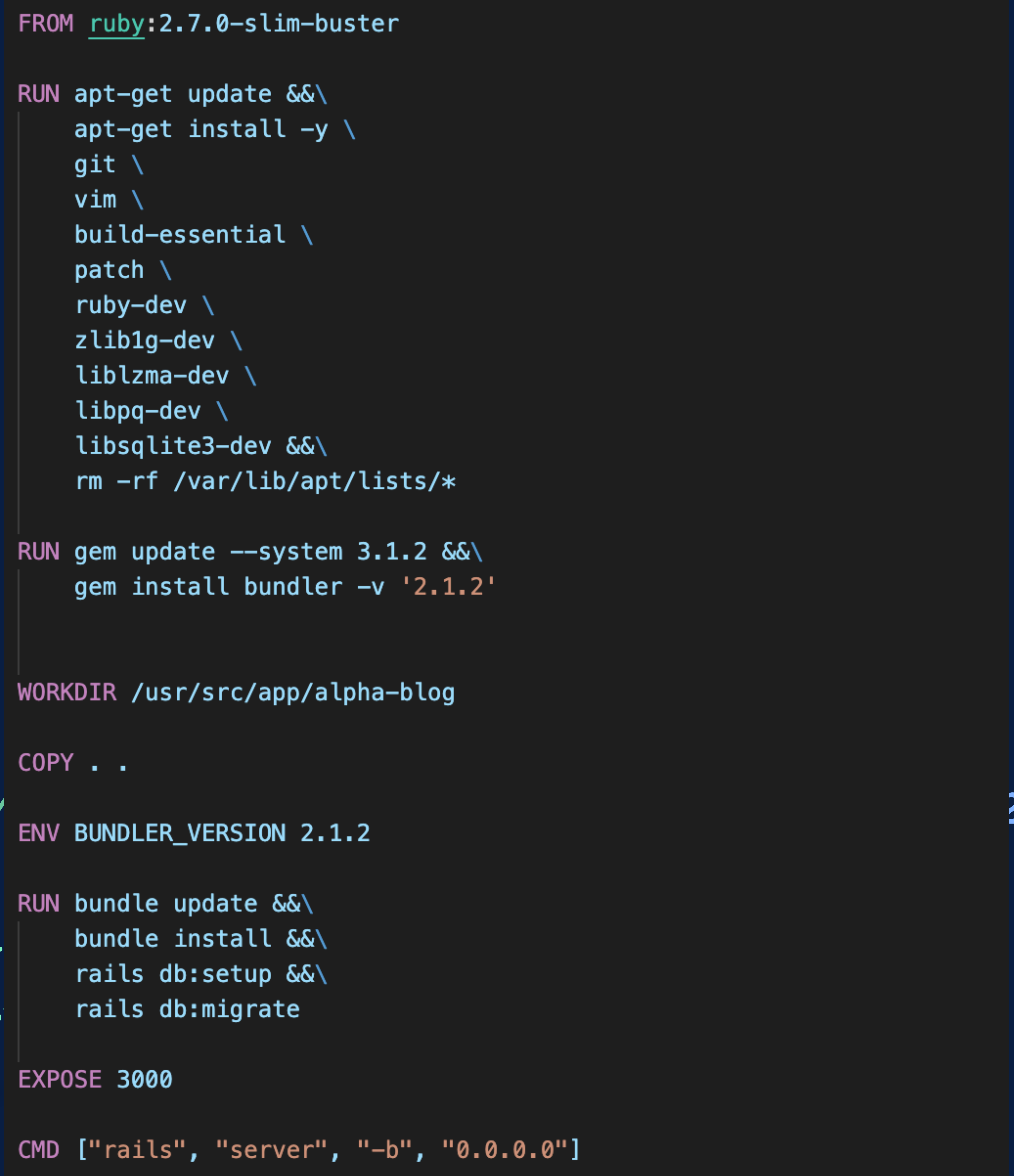
3-debian11 => 54.2MB

use specific packages (apt/yum/apk/...)

Base images:



```
FROM ruby:2.7.0

RUN apt-get update &&\
    apt-get install -y git vim sqlite3 &&\
    rm -rf /var/lib/apt/lists/*

RUN gem update --system 3.0.4 &&\
    gem install bundler -v '2.1.2'

WORKDIR /usr/src/app/alpha-blog

COPY . .

ENV BUNDLER_VERSION 2.1.2

RUN bundle update &&\
    bundle install &&\
    rails db:setup &&\
    rails db:migrate

EXPOSE 3000

CMD ["rails", "server", "-b", "0.0.0.0"]
```



```
FROM ruby:2.7.0-slim-buster

RUN apt-get update &&\
    apt-get install -y \
    git \
    vim \
    build-essential \
    patch \
    ruby-dev \
    zlib1g-dev \
    liblzma-dev \
    libpq-dev \
    libsqlite3-dev &&\
    rm -rf /var/lib/apt/lists/*

RUN gem update --system 3.1.2 &&\
    gem install bundler -v '2.1.2'


WORKDIR /usr/src/app/alpha-blog

COPY . .

ENV BUNDLER_VERSION 2.1.2

RUN bundle update &&\
    bundle install &&\
    rails db:setup &&\
    rails db:migrate

EXPOSE 3000

CMD ["rails", "server", "-b", "0.0.0.0"]
```

Use specifi

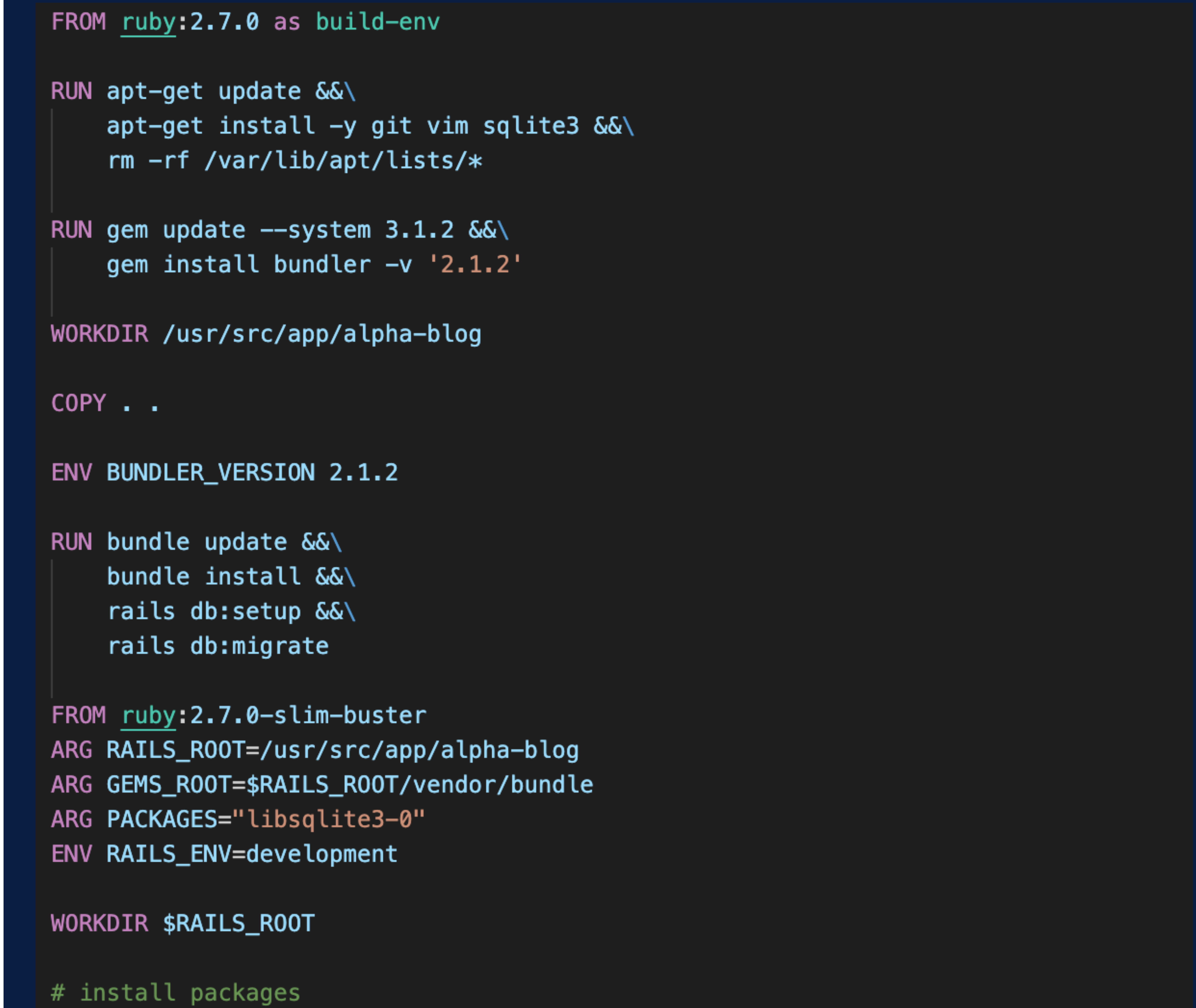
2MB

Vulnerable Containers

Image Architecture

Tools & Strategies

Supply Chain



```
FROM ruby:2.7.0 as build-env

RUN apt-get update &&\
    apt-get install -y git vim sqlite3 &&\
    rm -rf /var/lib/apt/lists/*

RUN gem update --system 3.1.2 &&\
    gem install bundler -v '2.1.2'

WORKDIR /usr/src/app/alpha-blog

COPY . .

ENV BUNDLER_VERSION 2.1.2

RUN bundle update &&\
    bundle install &&\
    rails db:setup &&\
    rails db:migrate

FROM ruby:2.7.0-slim-buster
ARG RAILS_ROOT=/usr/src/app/alpha-blog
ARG GEMS_ROOT=$RAILS_ROOT/vendor/bundle
ARG PACKAGES="libsqlite3-0"
ENV RAILS_ENV=development

WORKDIR $RAILS_ROOT

# install packages
```

Vulnerable Containers

Image Architecture

Tools & Strategies

Supply Chain

Multi-stage Builds:

- Multiple FROM statements
- Final stage = image that gets saved

Alternative build tools:

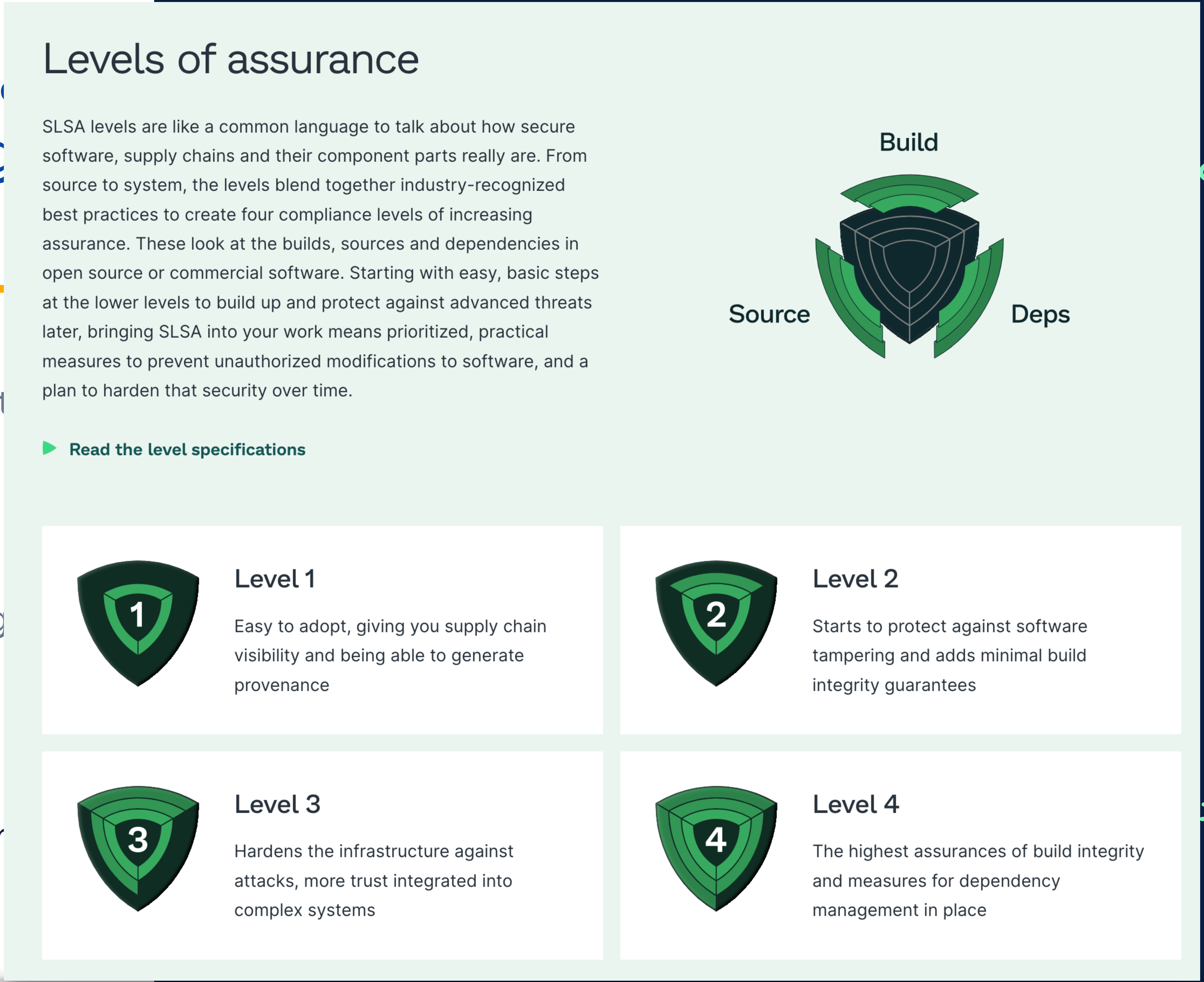
- kaniko
- jib
- ko
- buildah
- Kpack

Building
Running
Containe

Image Architect

Tools & Strateg

Supply Chai



Levels of assurance

SLSA levels are like a common language to talk about how secure software, supply chains and their component parts really are. From source to system, the levels blend together industry-recognized best practices to create four compliance levels of increasing assurance. These look at the builds, sources and dependencies in open source or commercial software. Starting with easy, basic steps at the lower levels to build up and protect against advanced threats later, bringing SLSA into your work means prioritized, practical measures to prevent unauthorized modifications to software, and a plan to harden that security over time.

▶ **Read the level specifications**

Level 1

Easy to adopt, giving you supply chain visibility and being able to generate provenance

Level 2

Starts to protect against software tampering and adds minimal build integrity guarantees

Level 3

Hardens the infrastructure against attacks, more trust integrated into complex systems

Level 4

The highest assurances of build integrity and measures for dependency management in place

ories

ifacts

Infrastructure as Code: Kubernetes

Resources

Understand Limits and Requests

SecurityContext

An API for securing pods and containers

NetworkPolicy

Built-in micro-segmented firewall

Policy Enforcement

Implement your policies as code

Infrastructure as Code i.e. Kubernetes

Resource limits & requests

SecurityContext:

runAsNonRoot, readOnlyRootFS, etc

Network Policies

Policy Enforcement

Container Resources

- Resource requests
    - Tells the scheduler how much resource is needed to start a container on.
    - Process may use more resources than the request specifies.

- Resource limits
    - Processes can only use up to the limits specified
    - Processes exceeding memory limits get "out of memory"
    - Processes exceeding CPU limits get throttled
    - Without setting limits, all of the node/host resources may be consumed which can be a DOS vector

Example 3: Infrastructure

Resource limits & requests

SecurityContext:

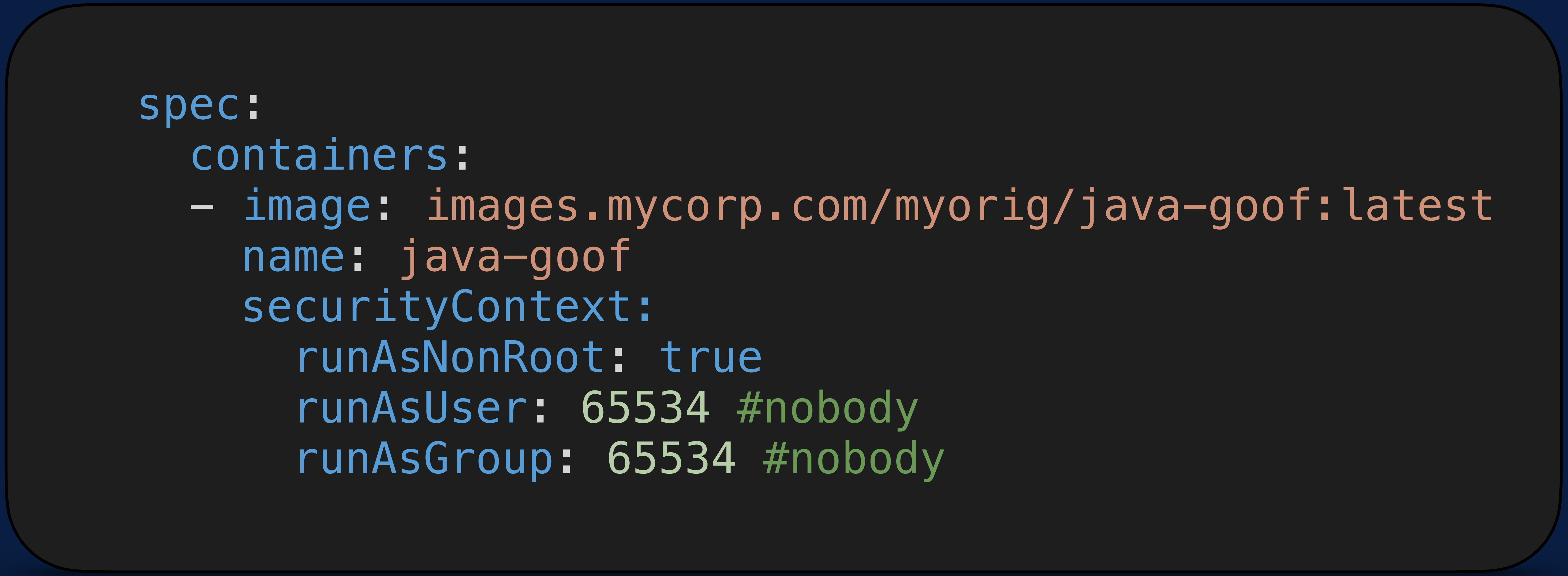
runAsNonRoot, readOnlyRootFS, etc

Network Policies

Policy Enforcement

Pod / Container SecurityContext API

- Run as non-root / Run as specific user
- Read-only root filesystem
- Linux capabilities
- Privileged mode
- Privilege Escalation



```yaml
spec:
  containers:
  - image: images.mycorp.com/myorig/java-goof:latest
    name: java-goof
    securityContext:
      runAsNonRoot: true
      runAsUser: 65534 #nobody
      runAsGroup: 65534 #nobody
```

Example 3: Infrastructure

Resource limits & requests

SecurityContext:

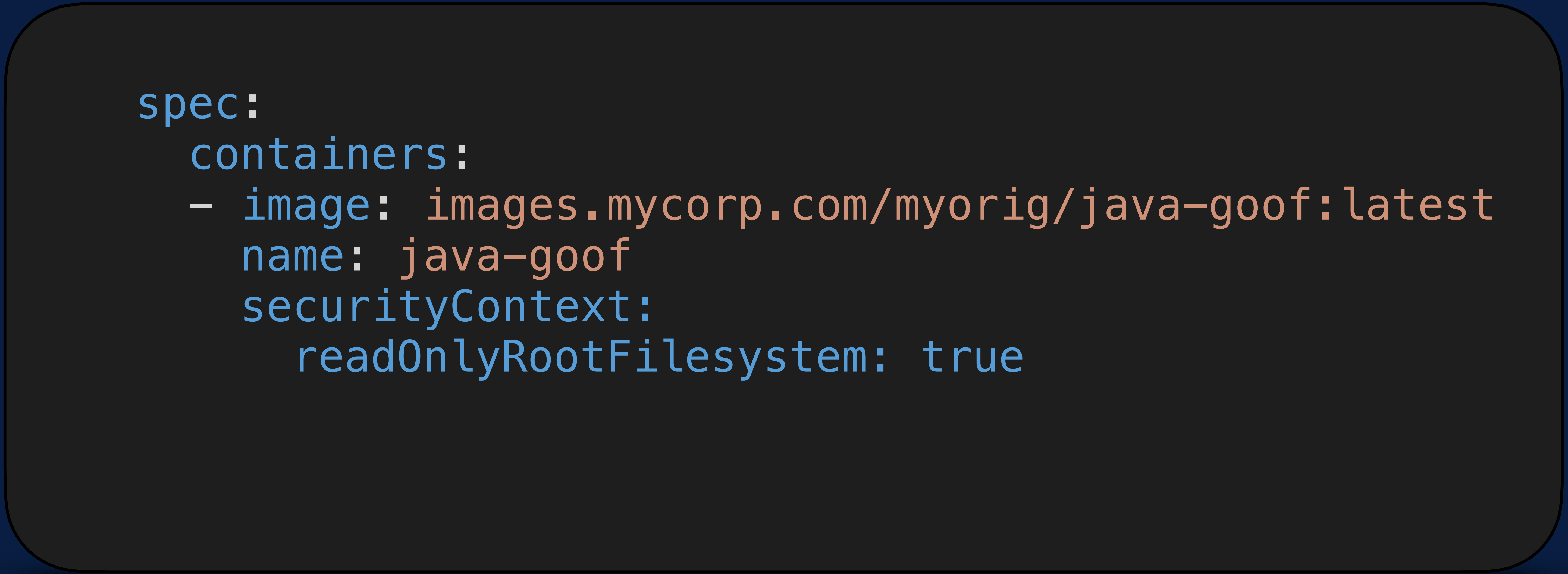
runAsNonRoot, readOnlyRootFS, etc

Network Policies

Policy Enforcement

Pod / Container SecurityContext API

- Run as non-root / Run as specific user
- Read-only root filesystem
- Linux capabilities
- Privileged mode
- Privilege Escalation



```yaml
spec:
  containers:
  - image: images.mycorp.com/myorig/java-goof:latest
    name: java-goof
    securityContext:
      readOnlyRootFilesystem: true
```

Example 3: Infrastructure

Resource limits & requests

SecurityContext:

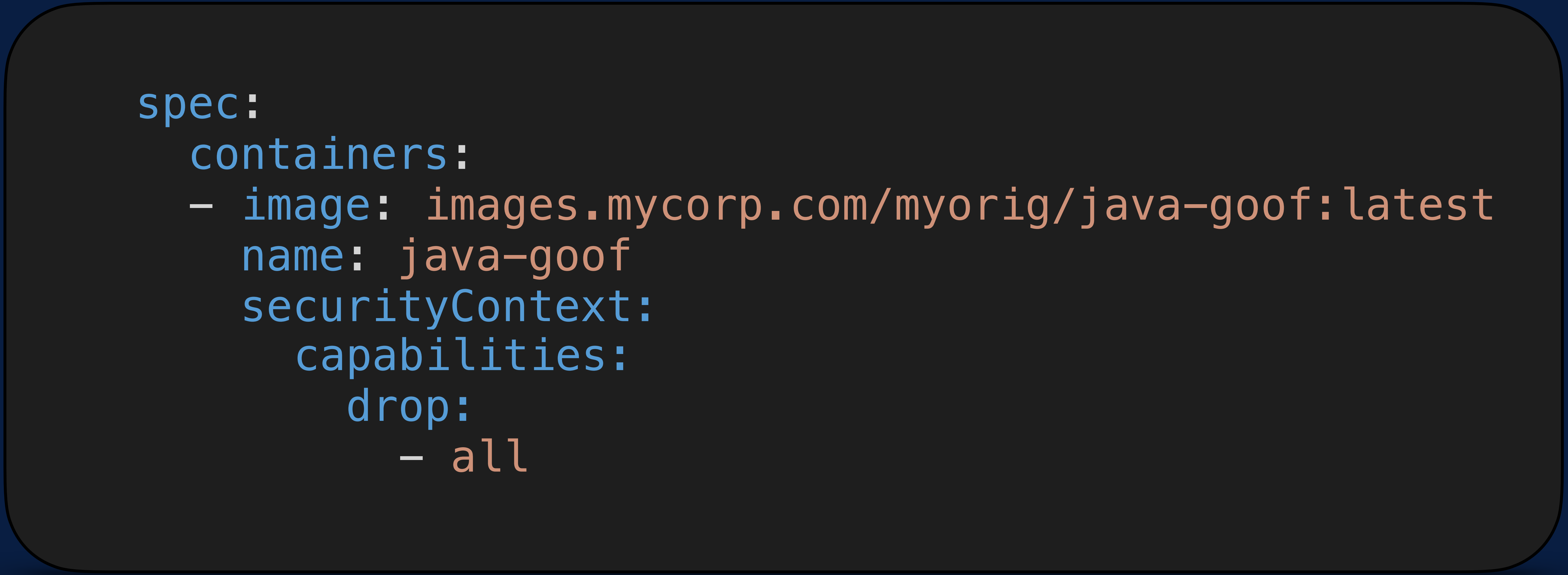
runAsNonRoot, readOnlyRootFS, etc

Network Policies

Policy Enforcement

Pod / Container SecurityContext API

- Run as non-root / Run as specific user
- Read-only root filesystem
- Linux capabilities
- Privileged mode
- Privilege Escalation



```
spec:
  containers:
  - image: images.mycorp.com/myorig/java-goof:latest
    name: java-goof
    securityContext:
      capabilities:
        drop:
          - all
```

Example 3: Infrastructure

Resource limits & requests

SecurityContext:

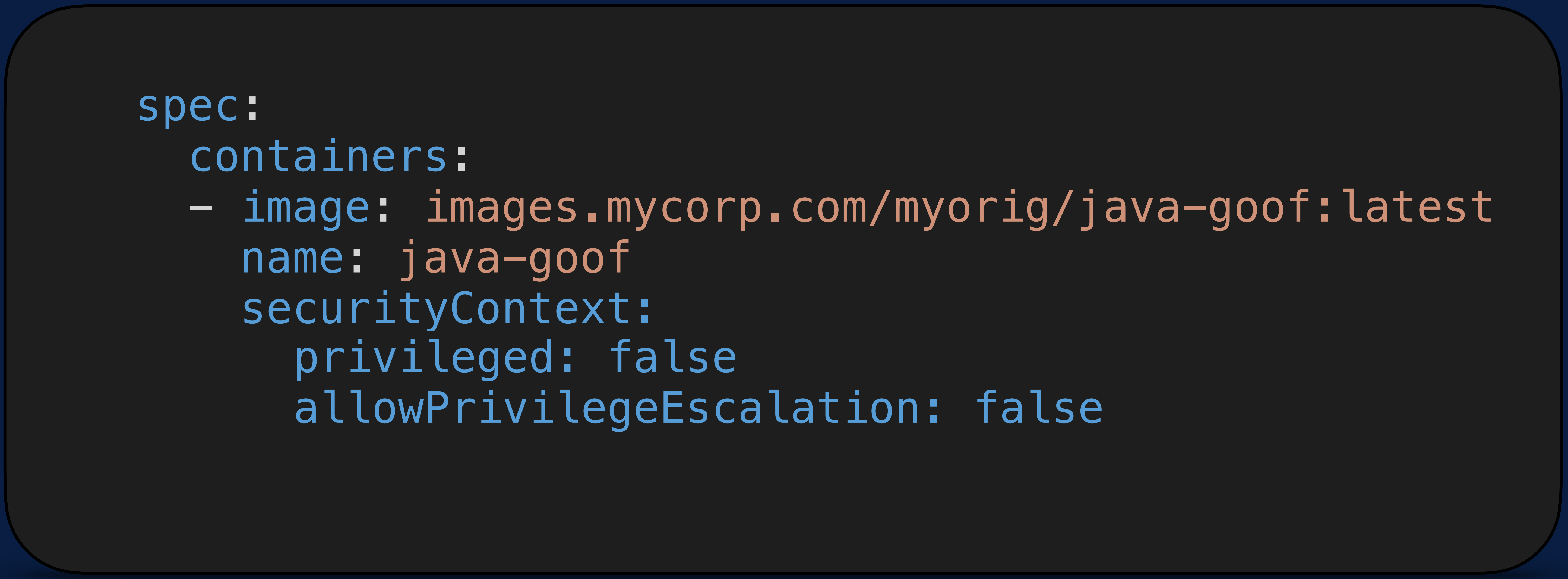
runAsNonRoot, readOnlyRootFS, etc

Network Policies

Policy Enforcement

Pod / Container SecurityContext API

- Run as non-root / Run as specific user
- Read-only root filesystem
- Linux capabilities
- Privileged mode
- Privilege Escalation



```
spec:
  containers:
  - image: images.mycorp.com/myorig/java-goof:latest
    name: java-goof
    securityContext:
      privileged: false
      allowPrivilegeEscalation: false
```

Example 3: Infrastructure

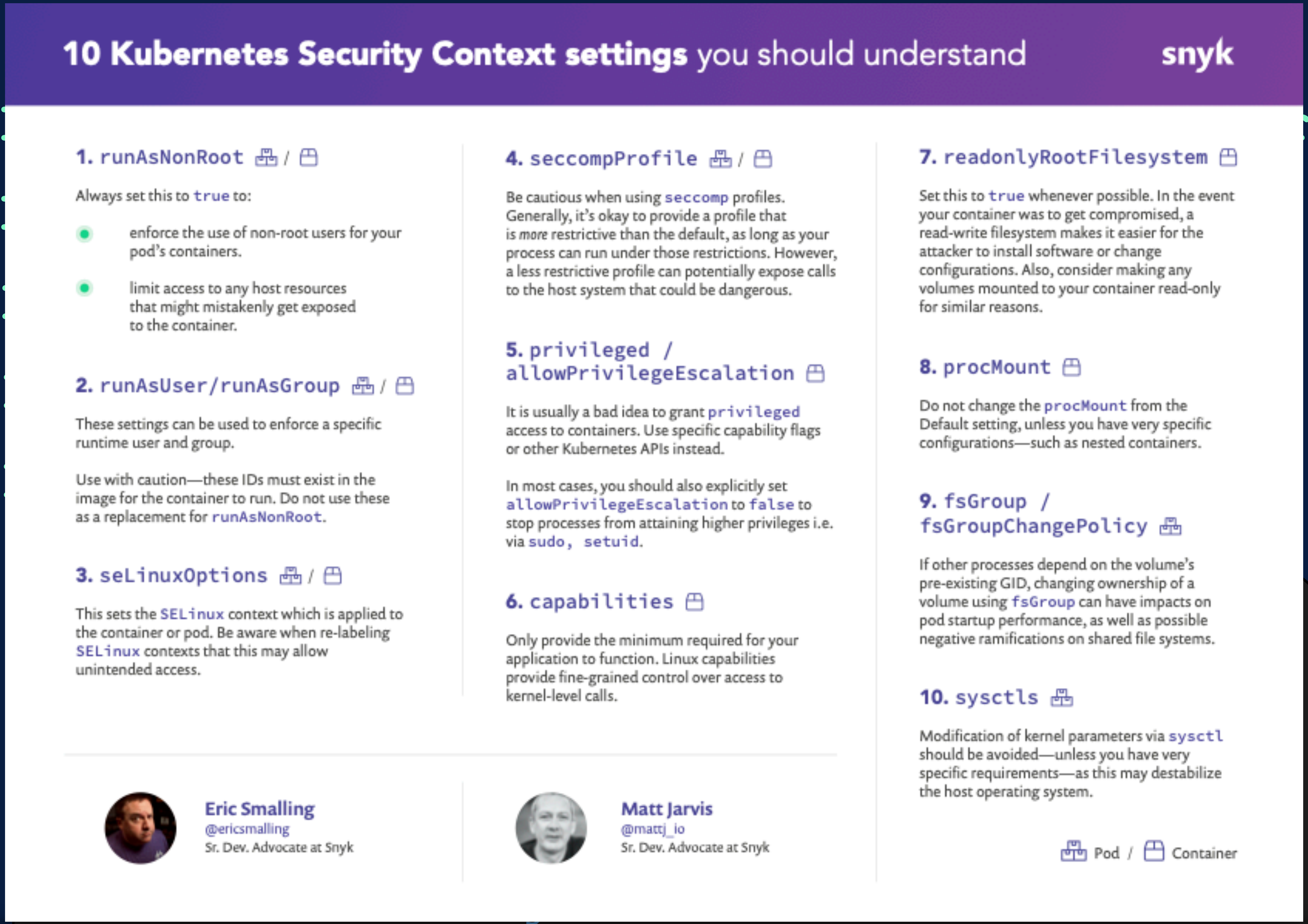
Resource limits & requests

SecurityContext:
runAsNonRoot,
readOnlyRootFS, etc

Network Policies

Policy Enforcement

Pod / Container SecurityContext API



10 Kubernetes Security Context settings you should understand   snyk

1. runAsNonRoot ⊞ / ⊟

Always set this to `true` to:

- enforce the use of non-root users for your pod's containers.
- limit access to any host resources that might mistakenly get exposed to the container.

2. runAsUser/runAsGroup ⊞ / ⊟

These settings can be used to enforce a specific runtime user and group.

Use with caution—these IDs must exist in the image for the container to run. Do not use these as a replacement for `runAsNonRoot`.

3. seLinuxOptions ⊞ / ⊟

This sets the `SELinux` context which is applied to the container or pod. Be aware when re-labeling `SELinux` contexts that this may allow unintended access.

4. seccompProfile ⊞ / ⊟

Be cautious when using `seccomp` profiles. Generally, it's okay to provide a profile that is *more* restrictive than the default, as long as your process can run under those restrictions. However, a less restrictive profile can potentially expose calls to the host system that could be dangerous.

5. privileged / allowPrivilegeEscalation ⊟

It is usually a bad idea to grant `privileged` access to containers. Use specific capability flags or other Kubernetes APIs instead.

In most cases, you should also explicitly set `allowPrivilegeEscalation` to `false` to stop processes from attaining higher privileges i.e. via `sudo, setuid`.

6. capabilities ⊟

Only provide the minimum required for your application to function. Linux capabilities provide fine-grained control over access to kernel-level calls.

7. readonlyRootFilesystem ⊟

Set this to `true` whenever possible. In the event your container was to get compromised, a read-write filesystem makes it easier for the attacker to install software or change configurations. Also, consider making any volumes mounted to your container read-only for similar reasons.

8. procMount ⊟

Do not change the `procMount` from the Default setting, unless you have very specific configurations—such as nested containers.

9. fsGroup / fsGroupChangePolicy ⊞

If other processes depend on the volume's pre-existing GID, changing ownership of a volume using `fsGroup` can have impacts on pod startup performance, as well as possible negative ramifications on shared file systems.

10. sysctls ⊞

Modification of kernel parameters via `sysctl` should be avoided—unless you have very specific requirements—as this may destabilize the host operating system.

Eric Smalling
@ericsmalling
Sr. Dev. Advocate at Snyk

Matt Jarvis
@mattj_io
Sr. Dev. Advocate at Snyk

⊞ Pod / ⊟ Container

https://snyk.co/uemWx

Example 3: Infrastructure

Resource limits & requests

SecurityContext:

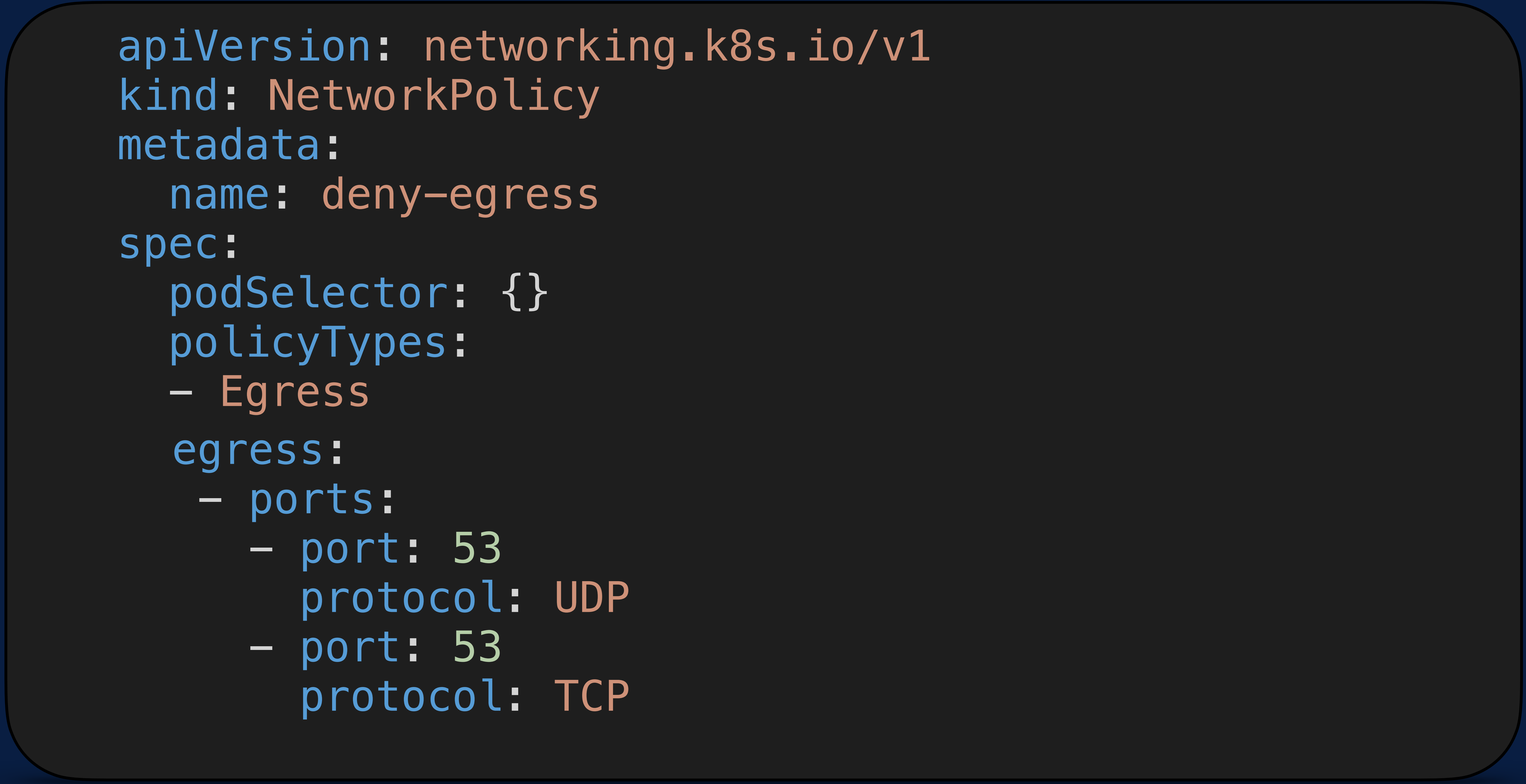
runAsNonRoot, readOnlyRootFS, etc

Network Policies

Policy Enforcement

Network Policies

- Pod-to-pod network traffic
- Micro-segmented firewall
- Deny-all policy
    - Limits unspecified ingress / egress



```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-egress
spec:
  podSelector: {}
  policyTypes:
  - Egress
  egress:
   - ports:
     - port: 53
       protocol: UDP
     - port: 53
       protocol: TCP
```

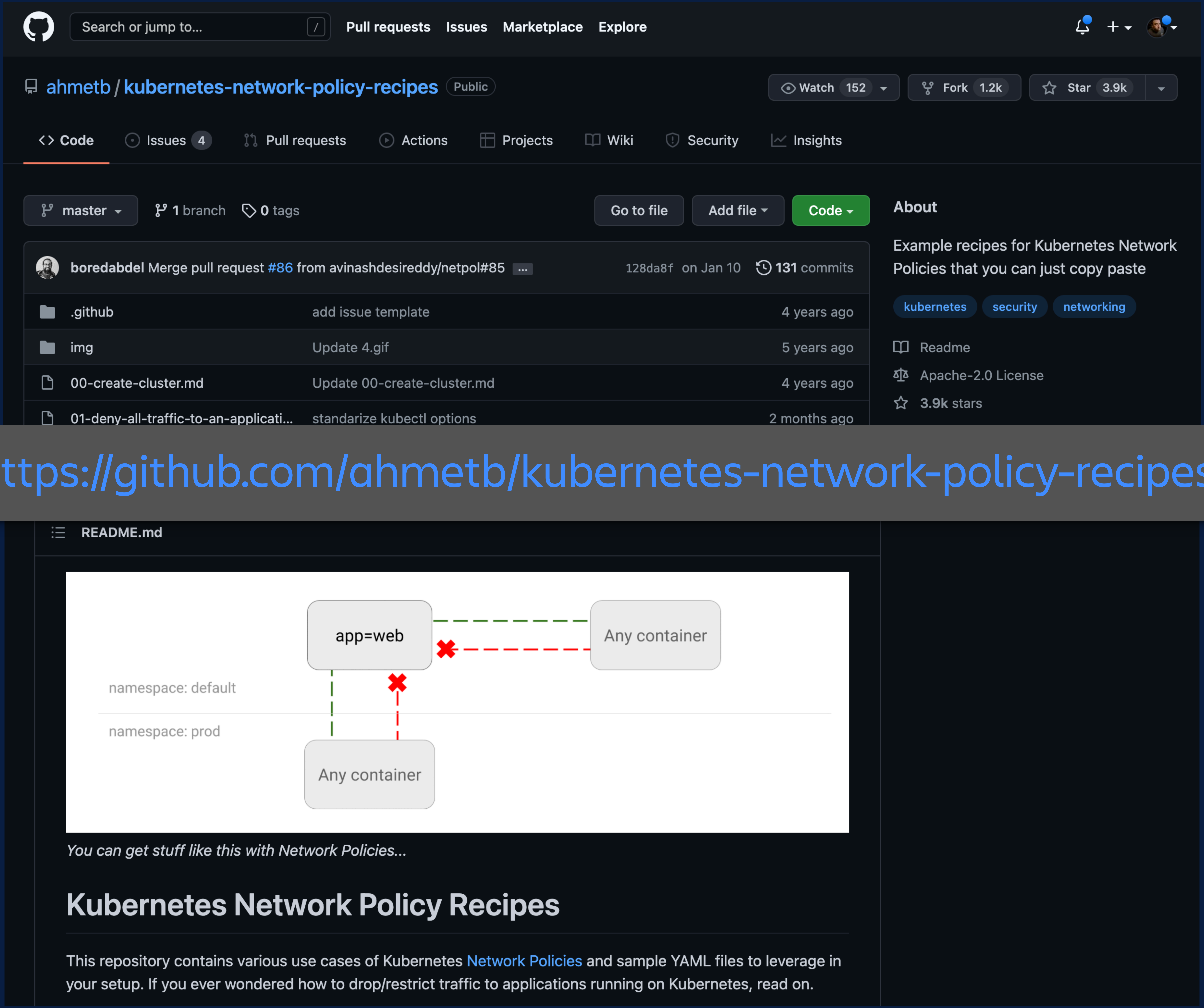
Example 3: Infrastructure

Resource limits & requests

SecurityContext: runAsNonRoot, readOnlyRootFS, etc

Network Policies

Policy Enforcement



ahmetb / kubernetes-network-policy-recipes Public

boredabdel Merge pull request #86 from avinashdesireddy/netpol#85 ... 128da8f on Jan 10 131 commits

.github	add issue template	4 years ago
img	Update 4.gif	5 years ago
00-create-cluster.md	Update 00-create-cluster.md	4 years ago
01-deny-all-traffic-to-an-applicati...	standarize kubectl options	2 months ago

About

Example recipes for Kubernetes Network Policies that you can just copy paste

kubernetes security networking

Readme

Apache-2.0 License

3.9k stars

https://github.com/ahmetb/kubernetes-network-policy-recipes

README.md

You can get stuff like this with Network Policies...

Kubernetes Network Policy Recipes

This repository contains various use cases of Kubernetes Network Policies and sample YAML files to leverage in your setup. If you ever wondered how to drop/restrict traffic to applications running on Kubernetes, read on.

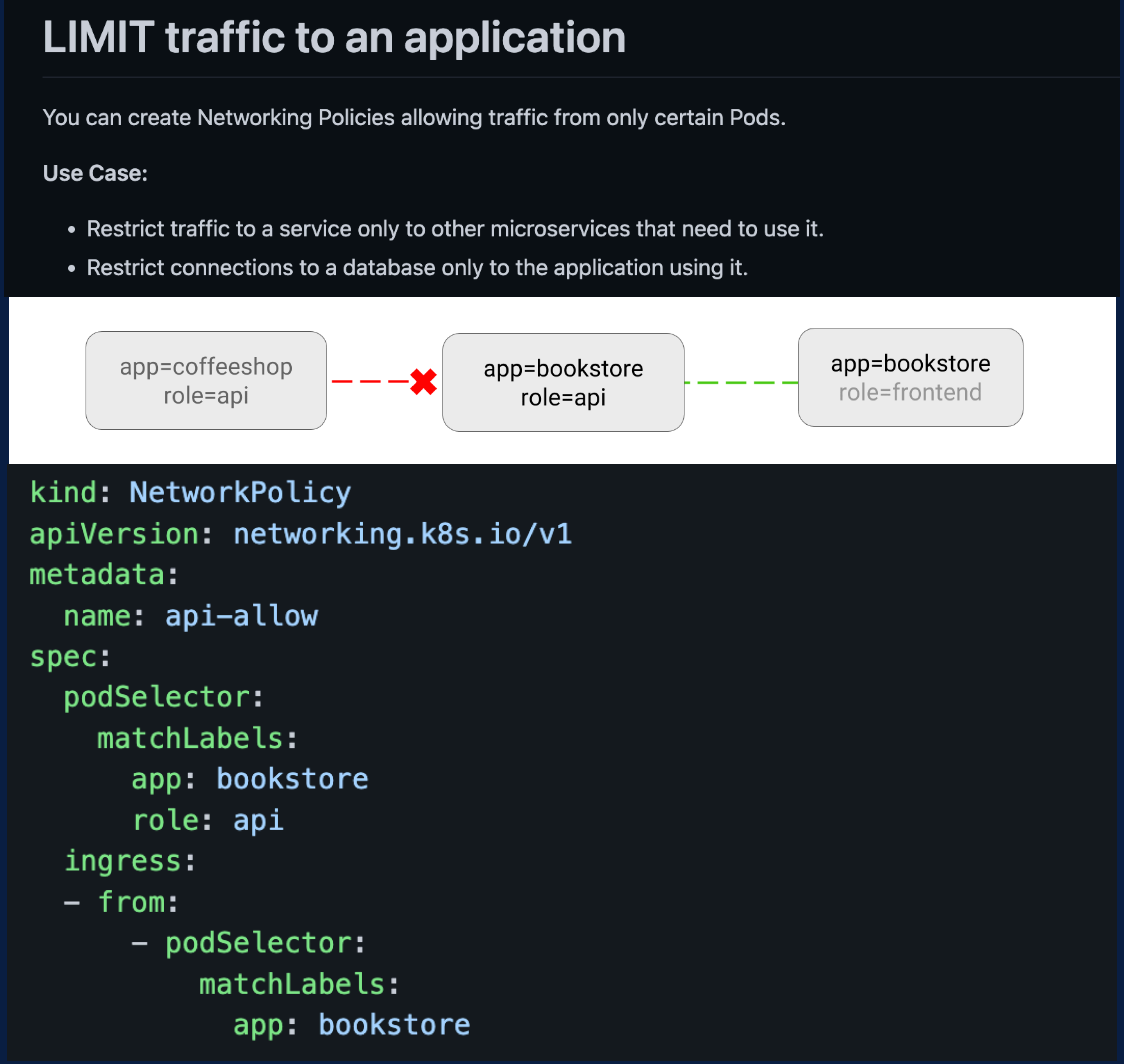
Example 3: Infrastructure

Resource limits & requests

SecurityContext:
runAsNonRoot,
readOnlyRootFS, etc

Network Policies

Policy Enforcement



LIMIT traffic to an application

You can create Networking Policies allowing traffic from only certain Pods.

Use Case:

- Restrict traffic to a service only to other microservices that need to use it.
- Restrict connections to a database only to the application using it.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: api-allow
spec:
  podSelector:
    matchLabels:
      app: bookstore
      role: api
  ingress:
  - from:
      - podSelector:
          matchLabels:
            app: bookstore
```

https://github.com/ahmetb/kubernetes-network-policy-recipes

Example 3: Infrastructure

Resource limits & requests

SecurityContext:

runAsNonRoot, readOnlyRootFS, etc

Network Policies

Policy Enforcement

```
Policy Enforcement Tools
- Pod Security Policy (PSP)
    - Deprecated v1.21
    - Removal in v1.25
- Pod Security Admission controller (PSA)
    - Replacement for PSP
    - Beta as of v1.23
    - Enforces Pod Security Standards
- Kyverno
    - Kubernetes specific
    - Policies defined in K8S CRDs
- OPA Gatekeeper
    - Open Policy Agent
    - Policies written in REGO
    - Gatekeeper admission controller
```

IN SUMMARY

Go get some small victories

One green light at a time
It is an iterative process

Work the multi-level / full stack

Code, Container, Infrastructure
Also the cloud/datacenter, networking, firewalls

Start now

Find a problem to solve, and work it

RESOURCES

Visit us

Snyk at Atlassian Team22: https://go.snyk.io/AtlassianTeamMeetings.html
Booth 2

Follow-us

@mrmarcoamorales

@marcoman

@ericsmalling

Links

Bitbucket Cheat Sheet: https://snyk.co/uemWw
Kubernetes Cheat Sheet: https://snyk.co/uemWx

ATLASSIAN

team'22

Thank you!