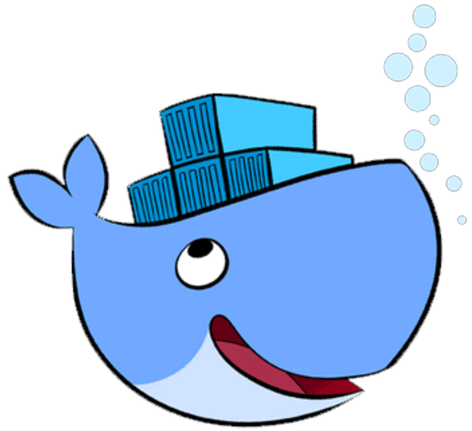
Container Patterns

Matthias Lübken

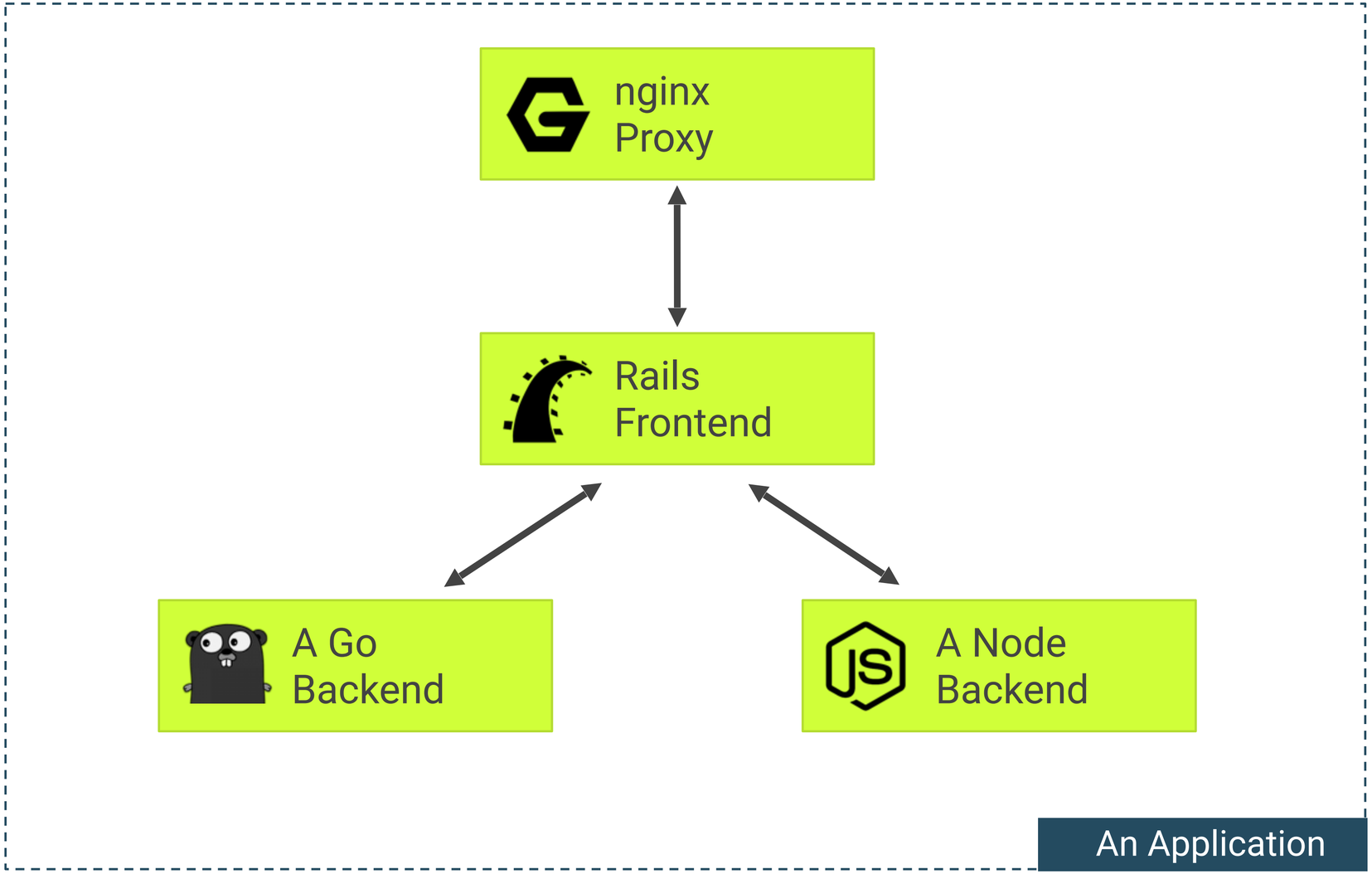
QCon LONDON

@luebken

“ Easily create lightweight,
portable, self-sufficient containers
from any **application**.”



“App Container (appc) is a well-specified and community developed specification for **application** containers. ”


nginx
Proxy
Rails
Frontend
A Go
Backend
A Node
Backend
An Application


Independently releasable
Separate processing types
Reuse
Different loads
Different teams
POCs
Crash isolation
Use different languages / versions / libraries

How does a good building block look like?

How do we assemble them?

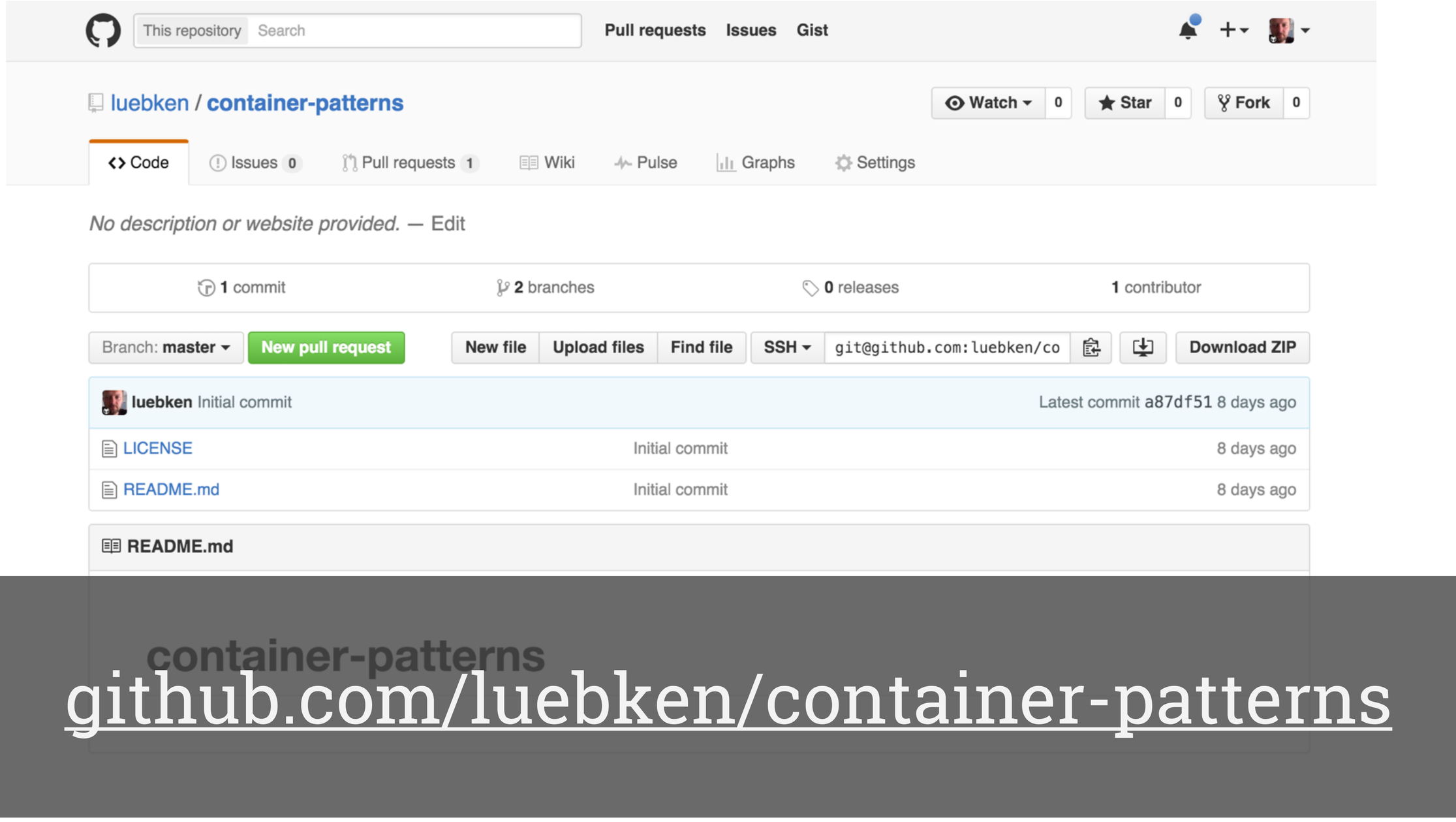
Related Work

- 12factor.net apps
- Cloud-native application architectures:
  Matt Stine Free Ebook
- Microservices
- Continuous Delivery

Container Patterns

- For designing “cloud” applications.
- Container **runtime agnostic**.
- Are there **general applicable** patterns?
- How would we **describe** them?
- What are concrete **examples** and best-practices?

I WANT YOU FOR AN ADVENTURE



github.com/luebken/container-patterns

How does a good building block look like?

How do we assemble them?

“Module-Container”

A “Module-Container” is a well behaving building block in the architecture of an application.

A Module Container is

1. Linux process >_
2. API
3. Descriptive
4. Disposable
5. Immutable
6. Self-contained
7. Small

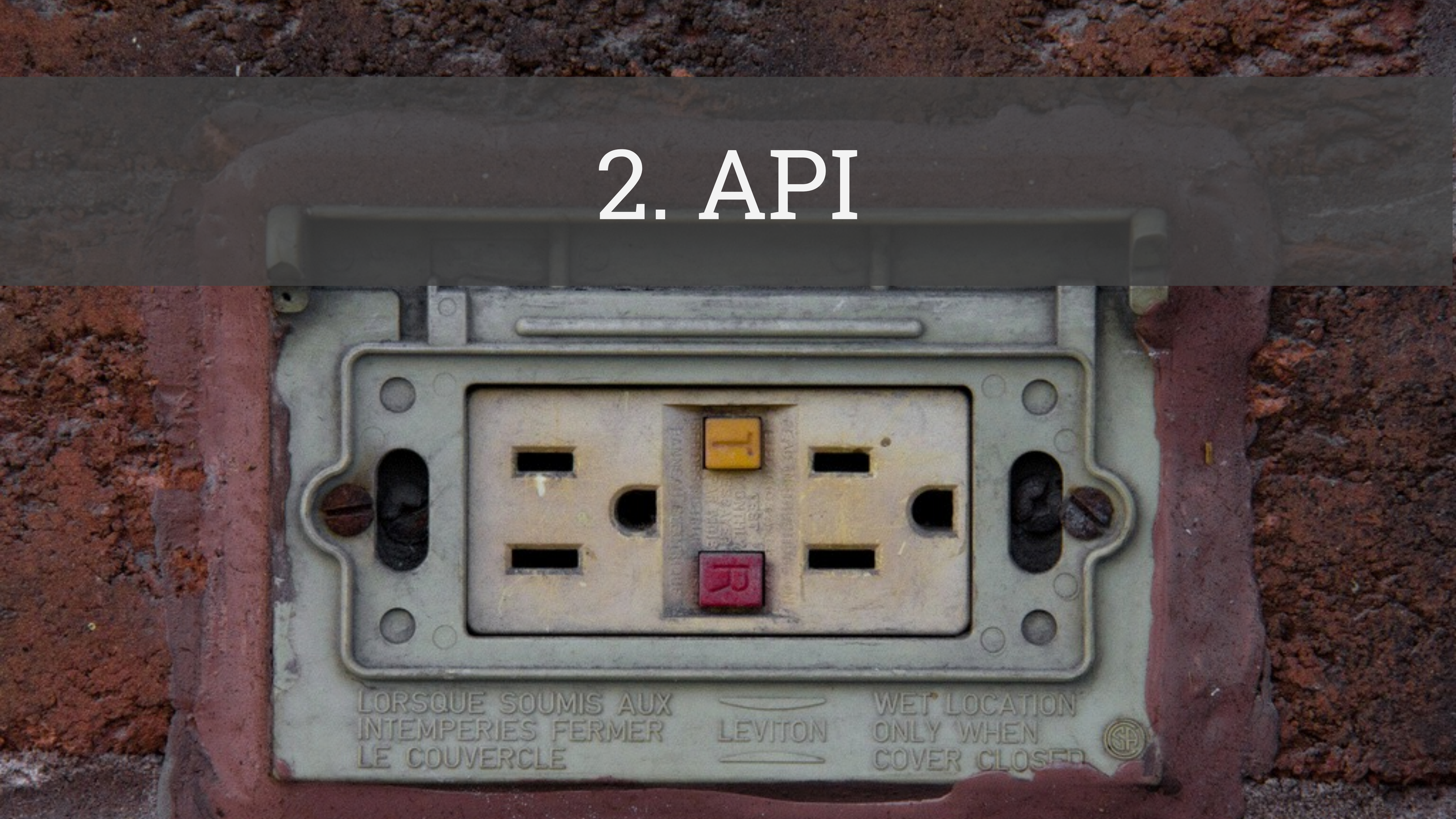
1. Linux Process

1. Linux Process

- React to signals
- Return exit codes
- Use standard streams
- Handle arguments

Examples: module-container.md#1-linux-process

2. API


LORSQUE SOUMIS AUX
INTEMPERIES FERMER
LE COUVERCLE
LEVITON
WET LOCATION
ONLY WHEN
COVER CLOSED

2. API

- ENV variables
- Available ports
- Volume mounts
- Lifecycle hooks

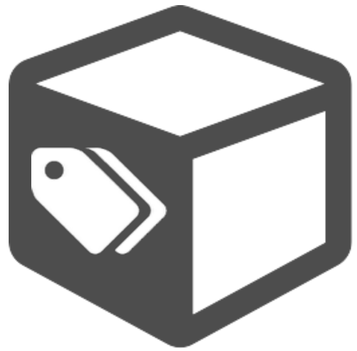
Examples: module-container.md#2-api

3. Descriptive

3. Descriptive

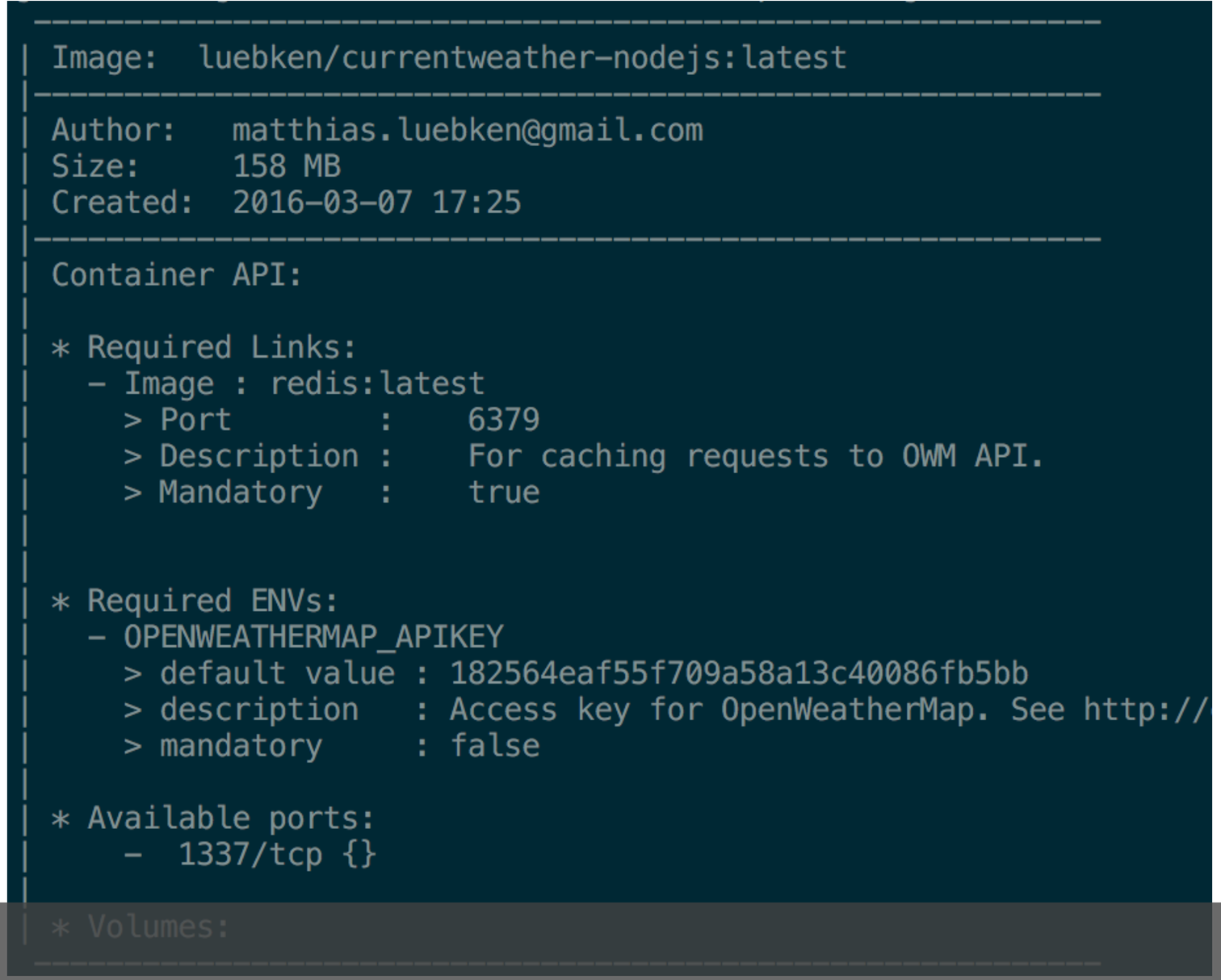
- Use standard labels (e.g. proposal generic labels)
  - url, summary, vcs-url …
- Use custom labels:
  - api.ENV
  - api.EXPOSE
  - api.LINKS

Examples: module-container.md#3-descriptive



Dockerfile

```
FROM node:0.10-slim
MAINTAINER matthias.luebken@gmail.com
WORKDIR /app

# install dependencies
ADD package.json /app/
RUN npm install

# install app
ADD server.js /app/

# Describe container dependencies
LABEL api.LINKS.redis=""\
      api.LINKS.redis.image="redis:latest"\
      api.LINKS.redis.port="6379"\
      api.LINKS.redis.description="For caching requests to OWM API."\
      api.LINKS.redis.mandatory="true"

# Set and describe available ENVs
ENV OPENWEATHERMAP_APIKEY=182564eaf55f709a58a13c40086fb5bb
LABEL api.ENV.OPENWEATHERMAP_APIKEY="" \
      api.ENV.OPENWEATHERMAP_APIKEY.description="Access key for OpenWeatherMap. See
      api.ENV.OPENWEATHERMAP_APIKEY.mandatory="false"

# Expose and describe available ports
EXPOSE 1337
LABEL api.EXPOSE.1337="" \
      api.EXPOSE.1337.protocol="http" \
      api.EXPOSE.1337.description="The main endpoint of this service."

ENTRYPOINT ["node", "server.js"]
```



POC github.com/luebken/container-api

4. Disposable

4. Disposable

- Don’t rely on a particular instance
- Be aware of shots at your cattle
- Be robust against sudden death

Examples: module-container.md#4-disposable


5. Immutable

5. Immutable

- Don’t change your container after build
- Strive or a dev/prod parity

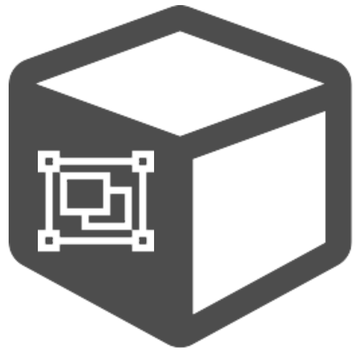
Examples: module-container.md#5-immutable


6. Self-contained

6. Self-contained

- Add dependencies on build time
- Sensible defaults

  Examples: module-container.md#6-self-contained



7. Small

7. Small

- Don’t use large base images
- Use the minimal footprint e.g. Alpine

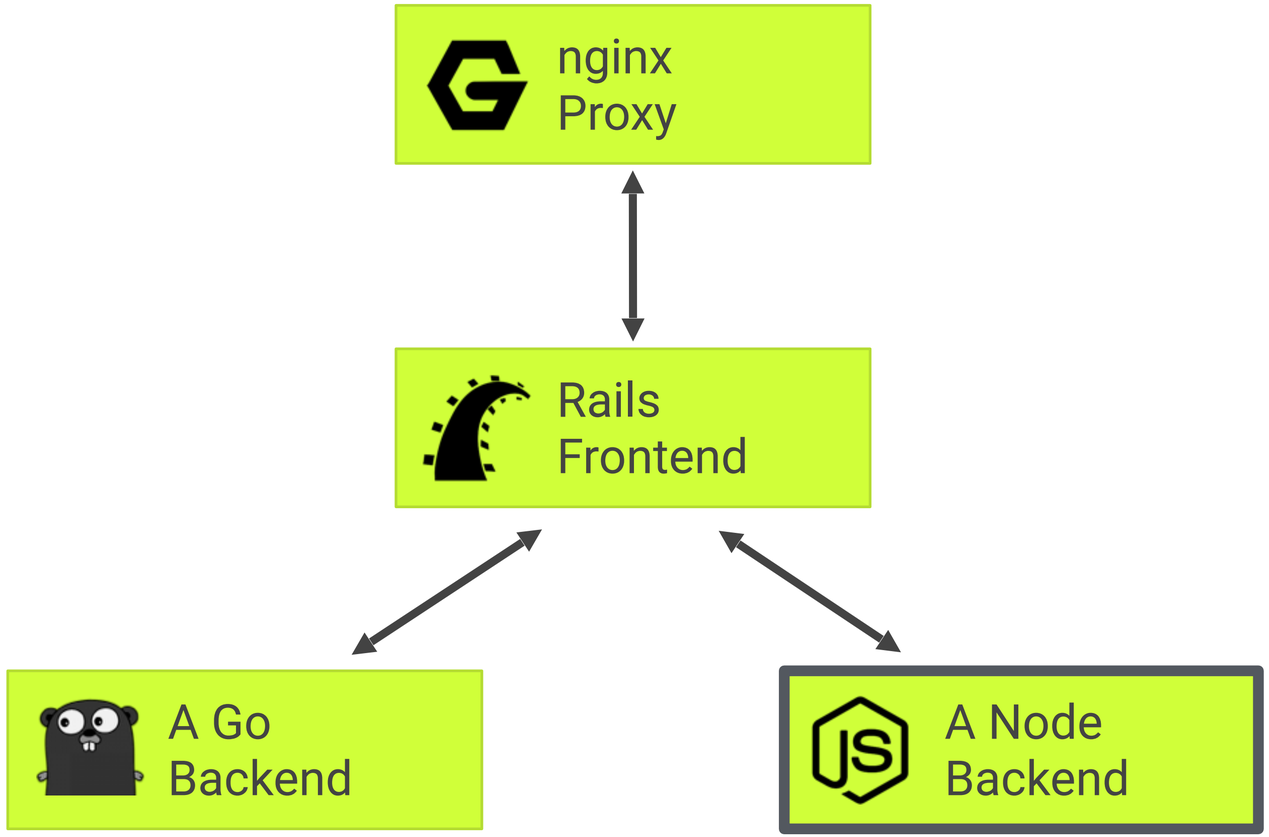
  Examples: module-container.md#6-small

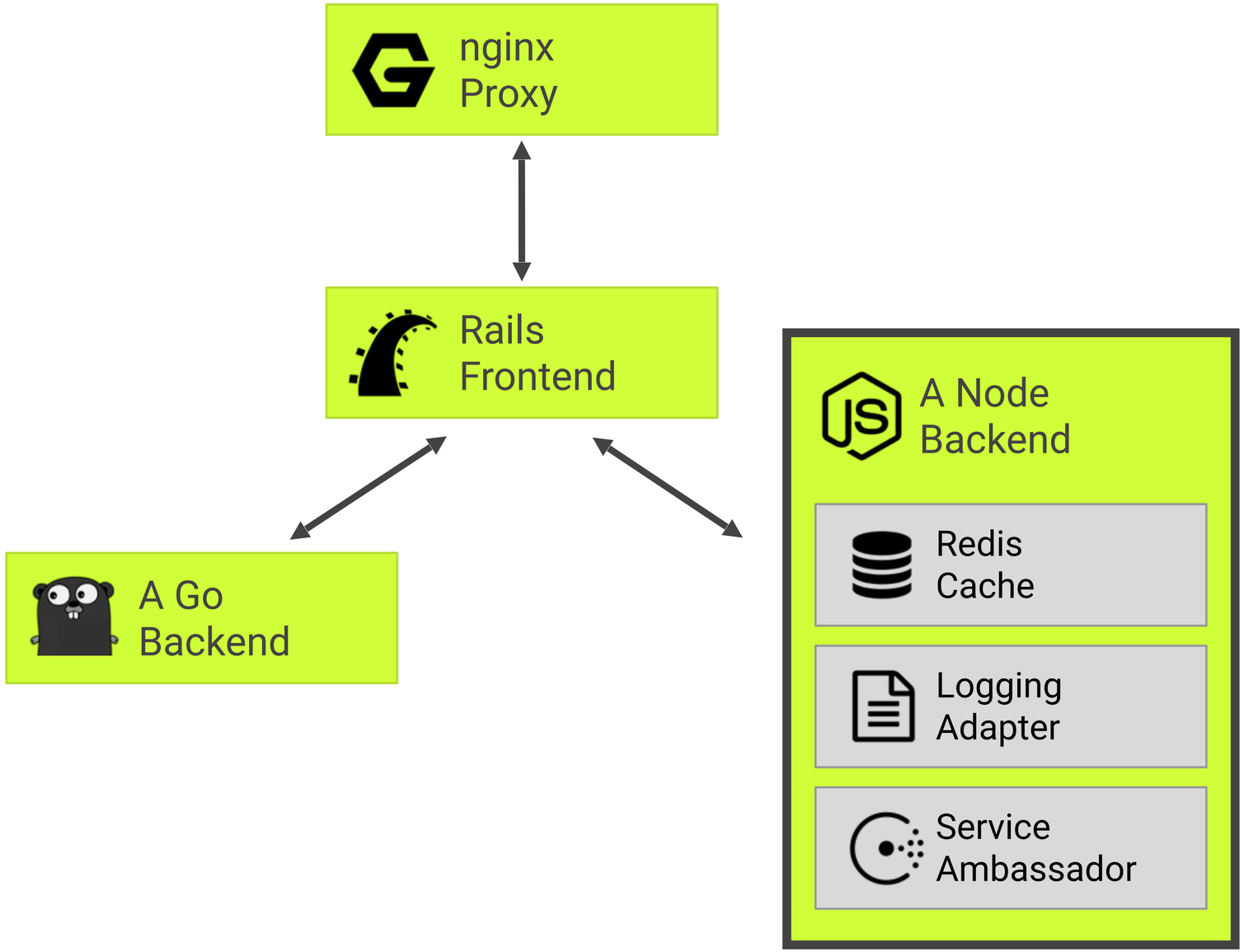
Recap: A Module Container is

1. Linux process >_
2. API
3. Descriptive
4. Disposable
5. Immutable
6. Self-contained
7. Small

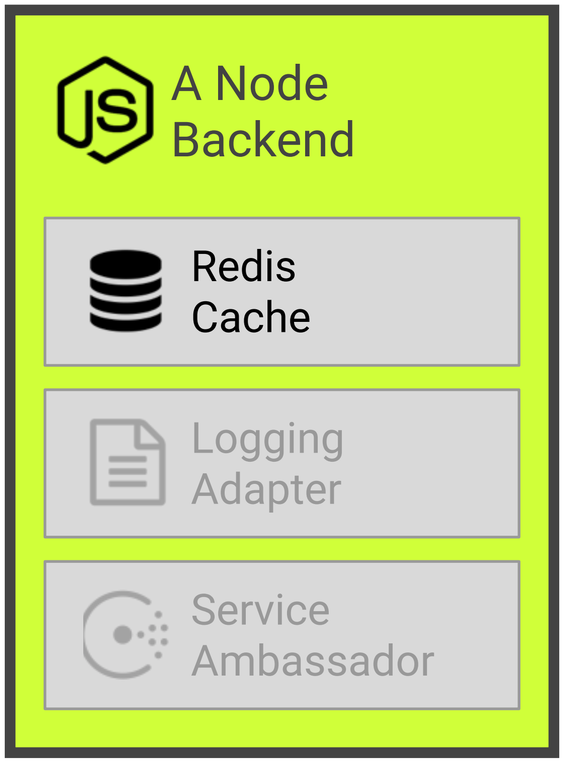
How does a good building block look like?

How do we assemble them?


nginx
Proxy
Rails
Frontend
A Go
Backend
A Node
Backend


nginx
Proxy
Rails
Frontend
A Go
Backend
A Node
Backend
Redis
Cache
Logging
Adapter
Service
Ambassador

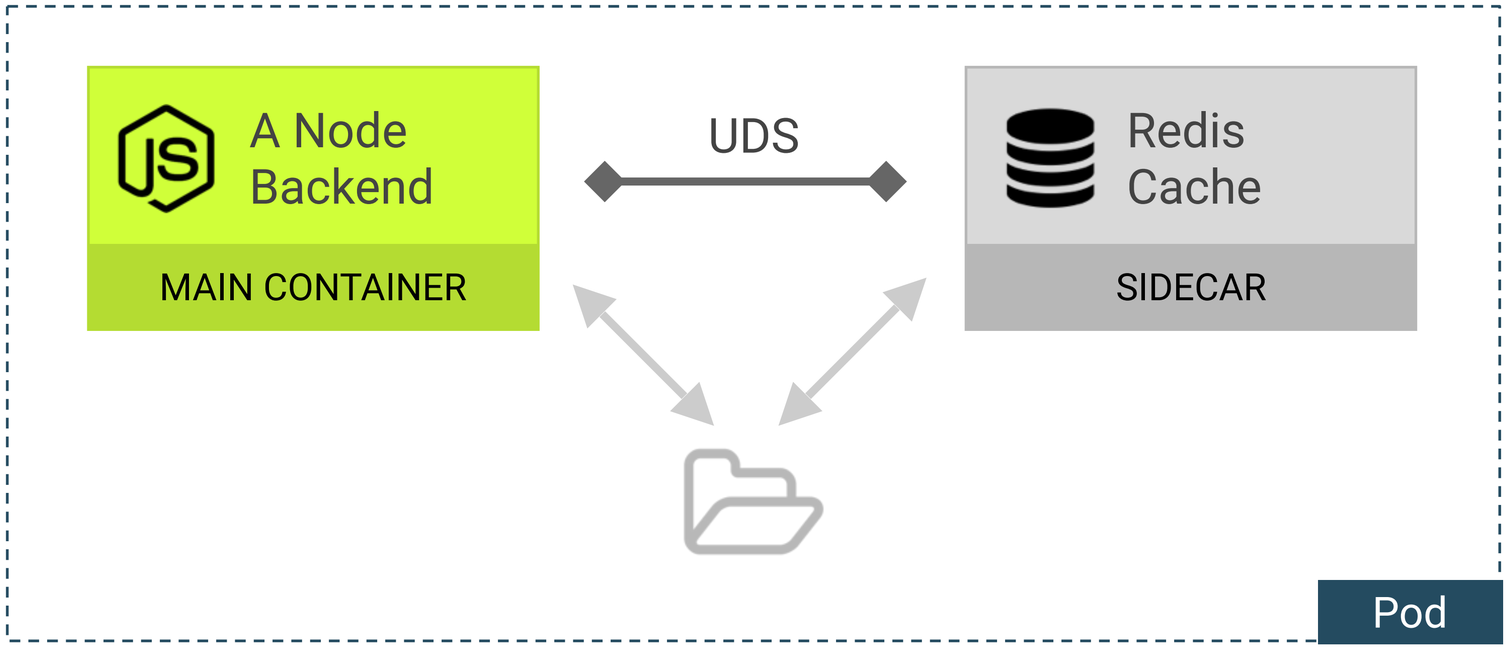
A group of closely related containers.
Deployed as a single unit
and share namespaces.

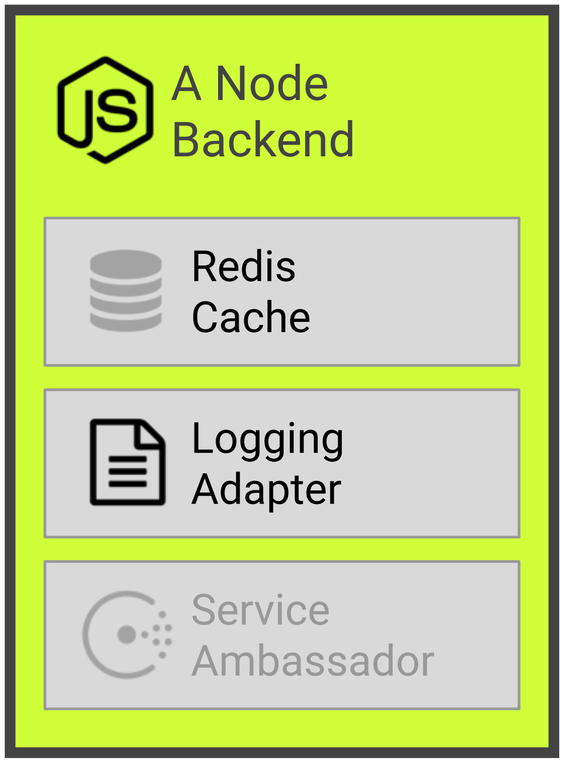

A Node
Backend
Redis
Cache
Logging
Adapter
Service
Ambassador

Pattern: Sidecar / Sidekick

Enhance & extend the main container.

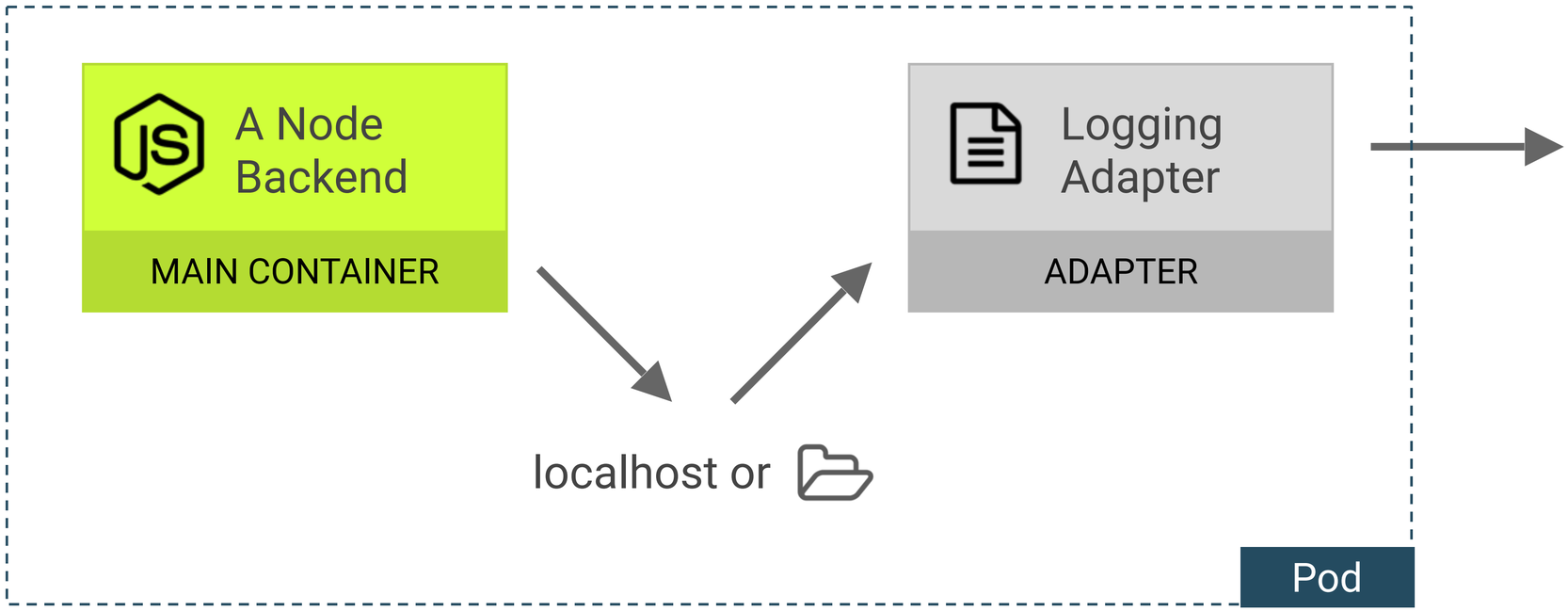
K8S: transparently. Netflix: platform features.

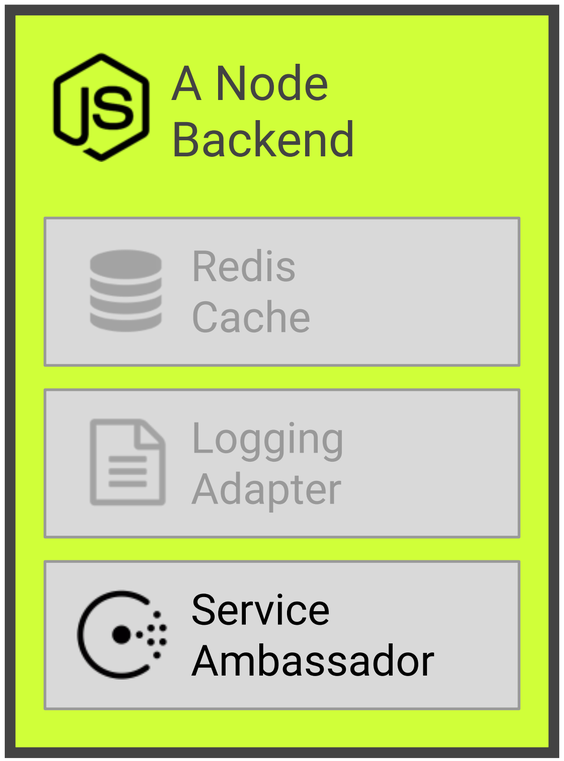



A Node
Backend
Redis
Cache
Logging
Adapter
Service
Ambassador

Pattern: Adapter

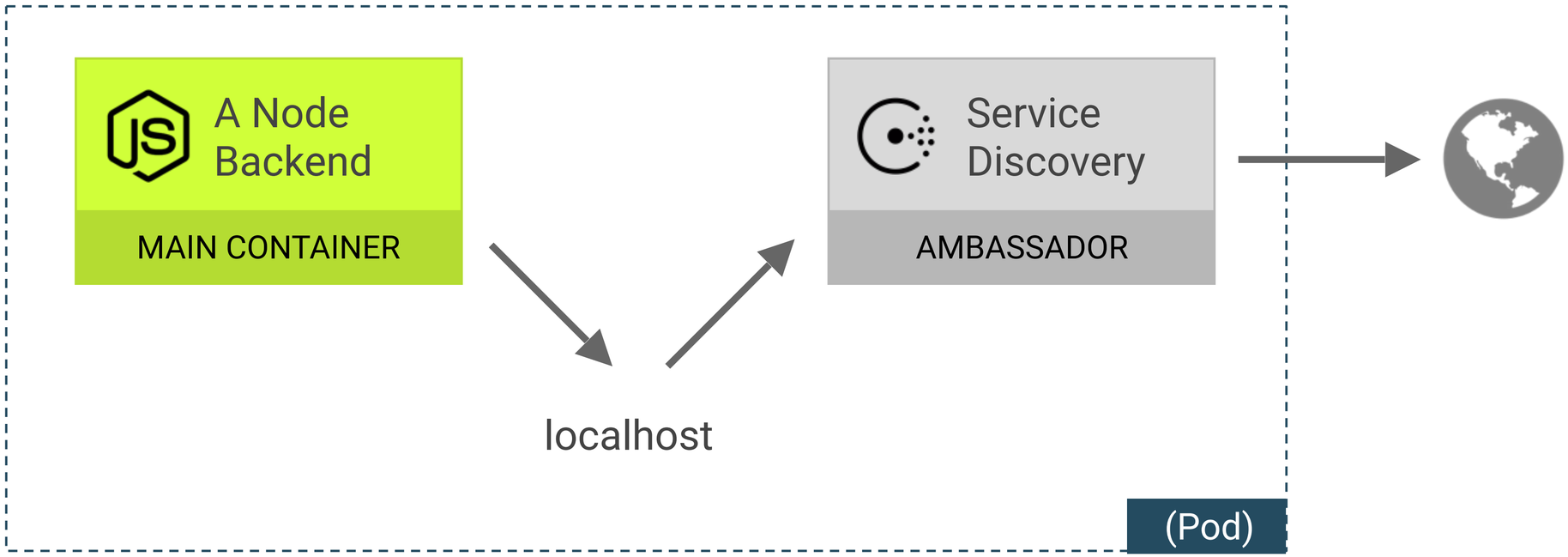
Standardise and normalize output.
E.g. logging and metrics.




A Node
Backend
Redis
Cache
Logging
Adapter
Service
Ambassador

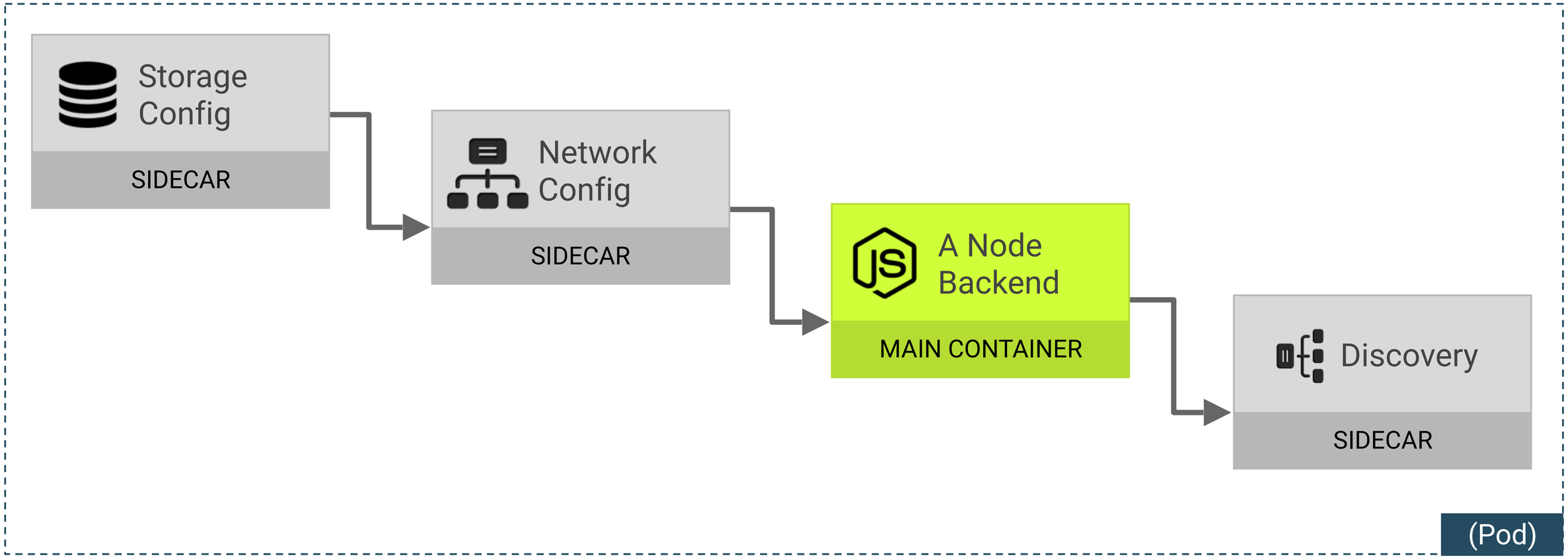
Pattern: Ambassador

Proxy a local connection to the world:
Service Discovery, Client Side LB, Circuit Breaker



Pattern: Container chains

Defined order of starting and stopping sidecar containers.



Summary: Container Patterns QCON 08.03.2016

How does a good building block look like?

And how would you assemble them?

Module container

1. Linux process
2. API
3. Descriptive
4. Disposable
5. Immutable
6. Self-contained
7. Small

Composite

- Sidecar
- Adapter
- Ambassador
- Chains

Dankeschön.

matthias@luebken.com

@luebken

Credits

- https://www.flickr.com/photos/skynoir/8241460998 (Cover image)
- https://www.flickr.com/photos/amlz/8664728590 (Lego)
- https://www.flickr.com/photos/guidedbycthulhu/6810361241 (Socket)
- https://www.flickr.com/photos/seektan/2074853585/ (Label)
- https://www.flickr.com/photos/gullevek/2122873934 (Trash)
- https://www.flickr.com/photos/grantmac/4852826923 (Lock)
- https://www.flickr.com/photos/mhirano/13236048424 (Hand)