

Cloud Native Non-Functional Requirements

Building Scalable, Resilient,
and Secure Applications

CLASSIFICATION: PUBLIC



Goals of This Talk



Explore relations

among principles, architectural approaches, requirements, and implementation



Provide practical insights

on how to build applications, which are not only functional – but built to last



Contents

Context

In which space are we working?

- Cloud Native
- NFRs
- Application Platforms

Approaches & Principles

Which qualities are important for us?

- Scalability
- Resilience
- Observability
- Security

Requirements & Implementation

How to achieve these qualities?

How Cloud Native Application Platforms help?

What is left for App Developers to implement?



Contents

Context

In which space are we working?

- Cloud Native
- NFRs
- Application Platforms

Approaches & Principles

Which qualities are important for us?

- Scalability
- Resilience
- Observability
- Security

Requirements & Implementation

How to achieve these qualities?

How Cloud Native Application Platforms help?

What is left for App Developers to implement?

Cloud Native

Cloud native technologies **empower** organizations to **build** and **run** scalable **applications** in modern, dynamic [cloud] environments [...]

These techniques enable **loosely coupled** systems that are **resilient**, **manageable**, and **observable**. Combined with robust **automation**, they allow engineers to make high-impact changes **frequently** and **predictably** with **minimal toil**.



Cloud Native

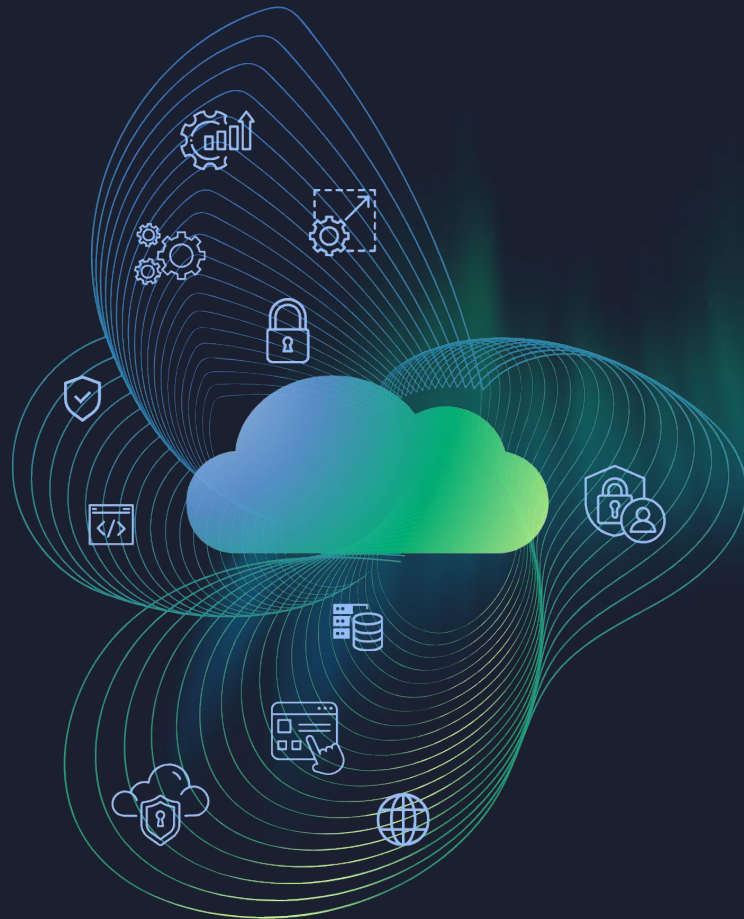


Non-Functional Requirements

NFRs are **system qualities** that **guide** the **design** of the solution and often **serve as constraints** [...]

NFRs **specify** various system **qualities and attributes**, such as:

- Performance
- Scalability
- Security
- Usability
- Maintainability



Application Platform

A comprehensive **set of tools and services** that **streamline** the entire **application lifecycle**, from **development** to **delivery** to **management** of app workloads.





Contents

Context

In which space are we working?

- Cloud Native
- NFRs
- Application Platforms

Approaches & Principles

Which qualities are important for us?

- Scalability
- Resilience
- Observability
- Security

Requirements & Implementation

How to achieve these qualities?

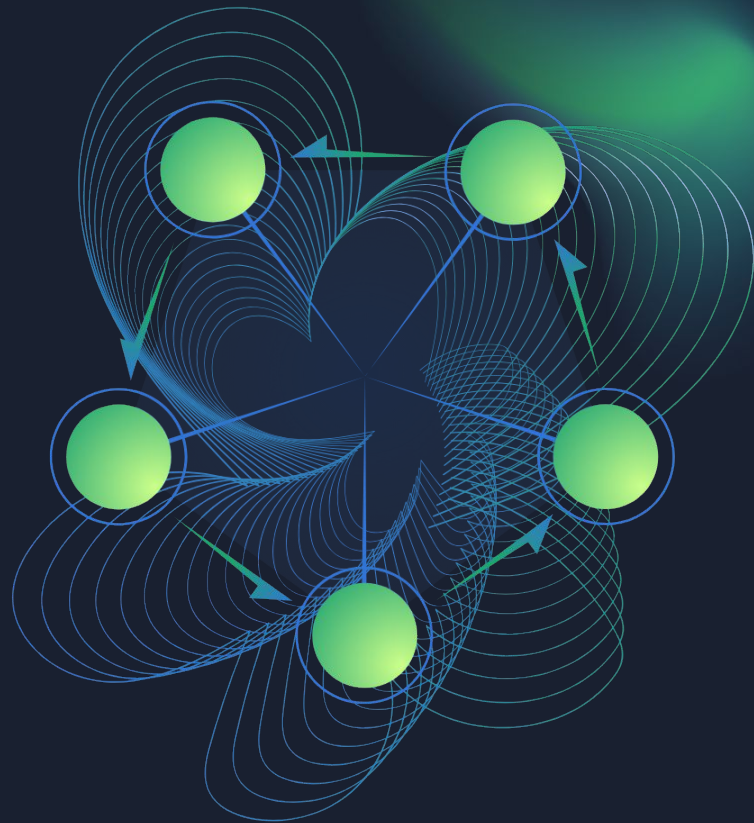
How Cloud Native Application Platforms help?

What is left for App Developers to implement?



Loose Coupling

- Architectural style
- Independent components
- Each **microservice**
 - has a specific function
 - can be used by any other
- **Drawback:** Slower to implement
- **Benefits:** Scales better since it allows to be independently
 - developed
 - deployed
 - scaled



Observability

- Generate actionable insights
- Understand a system's state

Benefits:

- Achieve favorable outcomes
 - Faster incident response
 - Increased dev productivity
 - Less toil and downtime
- Impacts operating and development costs



Automation

Continuous Integration (CI)

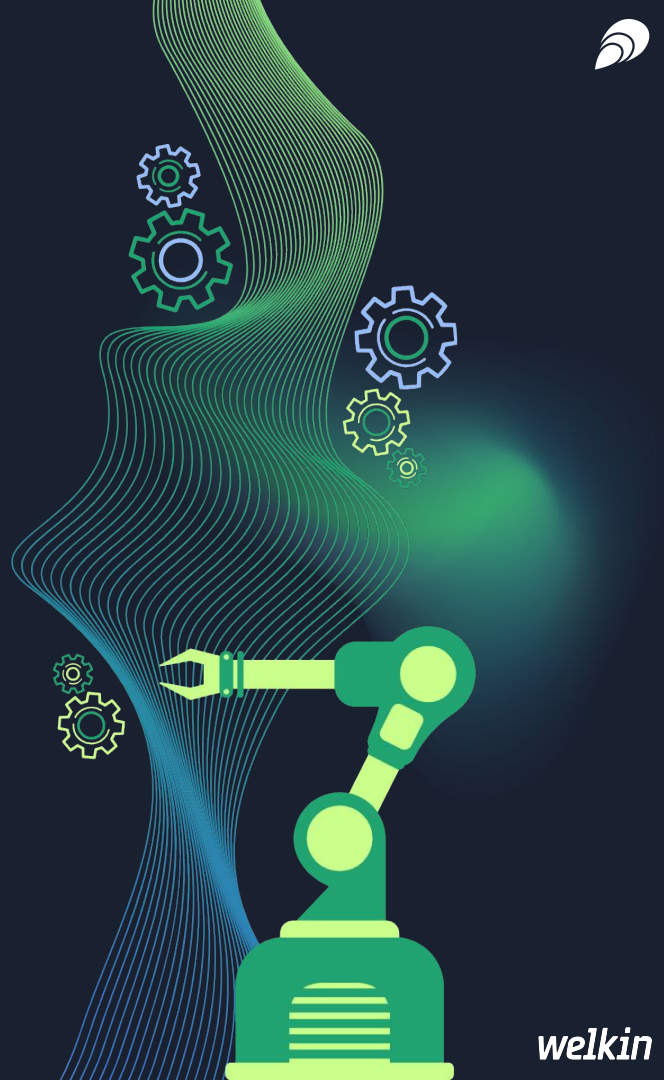
- Support parallel work
- Code quality checks, tests
- Each change → a release candidate

Continuous Delivery (CD)

- Deploy to acceptance environment
- Reliable updates at high frequency

Continuous Deployment (CD)

- Deploy to prod (Canary, Blue/Green)
- Automatic rollbacks
- Eliminate painful and error-prone manual steps



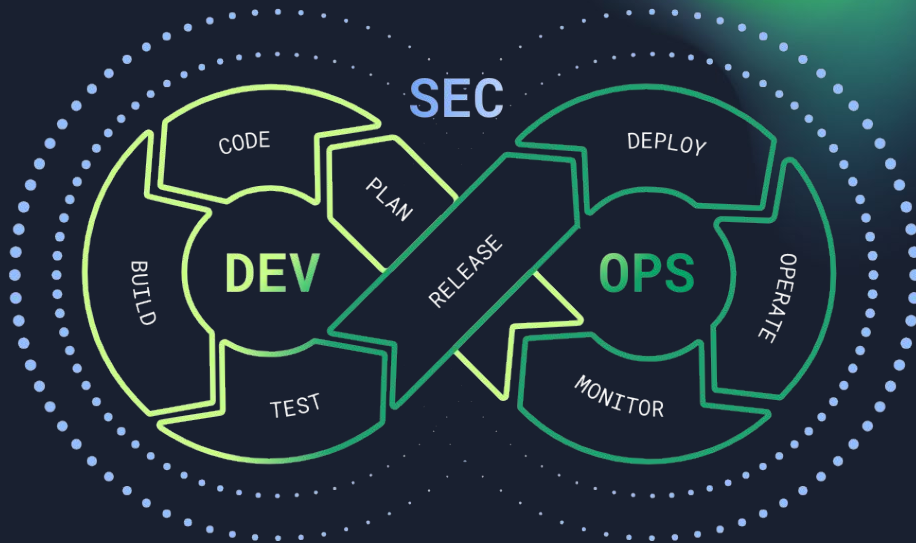
Security

- Security is now part of the **entire application lifecycle**
- Ensure **highest standards** while **adapting** to cloud native reality
 - Rapid code changes
 - Highly ephemeral infrastructure



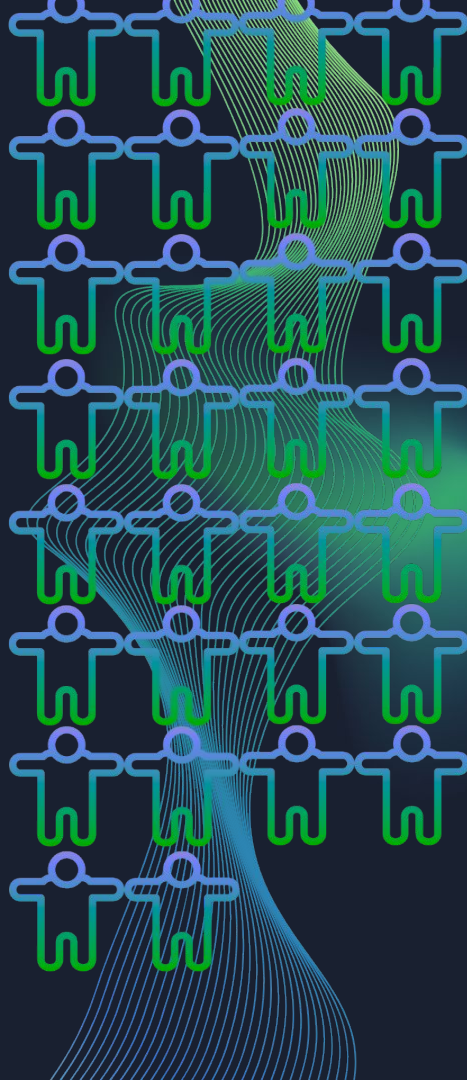
Security Principles

- ✓ Security by Design
- ✓ Minimal Need to Know
- ✓ Defence in Depth
- ✓ Redundancy



Don't Repeat Yourself

- Principle of software development
- “Duplication is evil”
- Eliminate:
 - in logic via abstraction
 - in process via automation
- “Control Technical Diversity”
- Reuse functionality offered by the application platform





Contents

Context

In which space are we working?

- Cloud Native
- NFRs
- Application Platforms

Approaches & Principles

Which qualities are important for us?

- Scalability
- Resilience
- Observability
- Security

Requirements & Implementation

How to achieve these qualities?

How Cloud Native Application Platforms help?

What is left for App Developers to implement?



Category of NFRs

Provided by Platform

✓ What you get “out of the box”?

Implemented by App Devs

✓ What you can focus on to provide unique value?



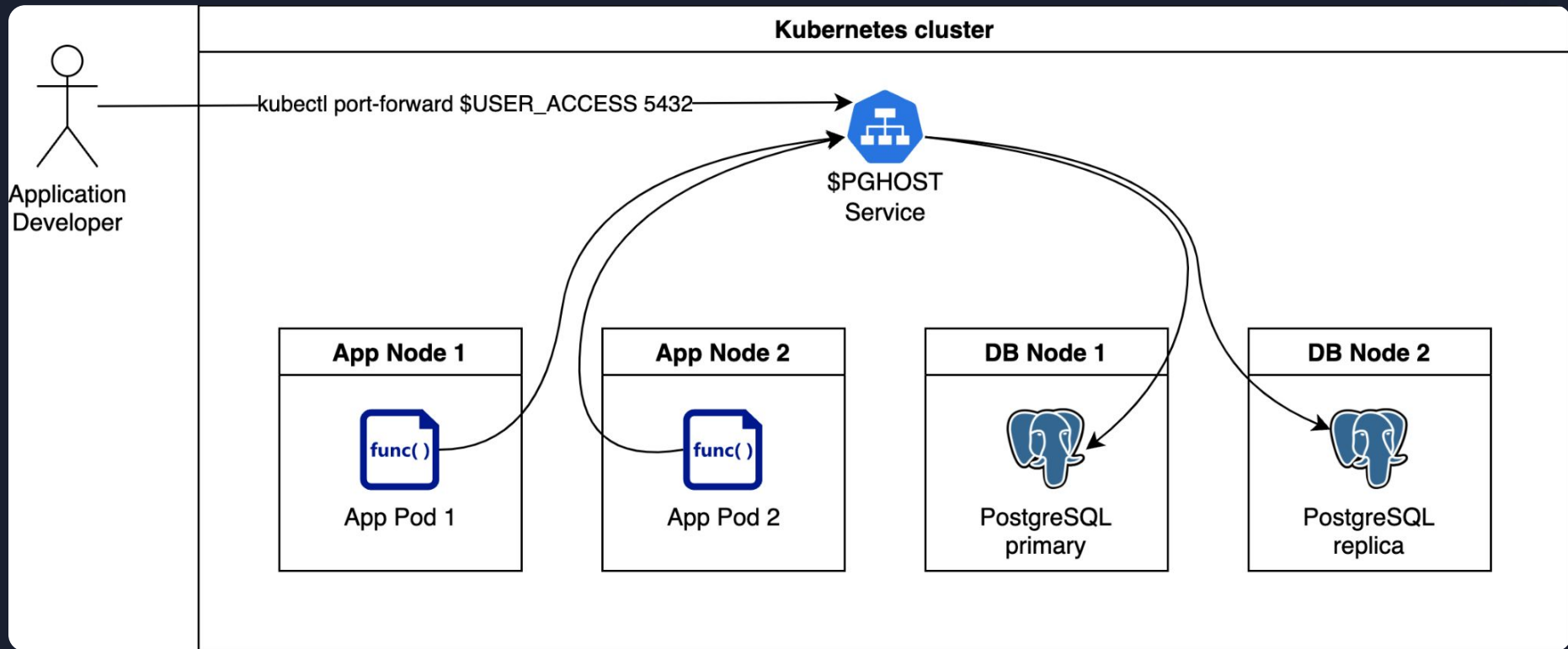
State management – Database

Provided by Platform

- ✓ Replicated setup ensuring business continuity
- ✓ Backups for disaster recovery
- ✓ Security patching

Implemented by App Devs

- ✓ Use DBS provided by a platform
- ✓ Manage application databases & users
- ✓ Expose credentials to your application
- ✓ Implement database migrations
 - ✓ Backwards-compatible
 - ✓ Rollbacks
 - ✓ Using init containers





Observability

Provided by Platform

- ✓ Normalization, filtering processing and shipping
- ✓ Log rotation
- ✓ Searching
- ✓ Visualization
- ✓ Alerting
- ✓ Protection of Log Information

Implemented by App Devs

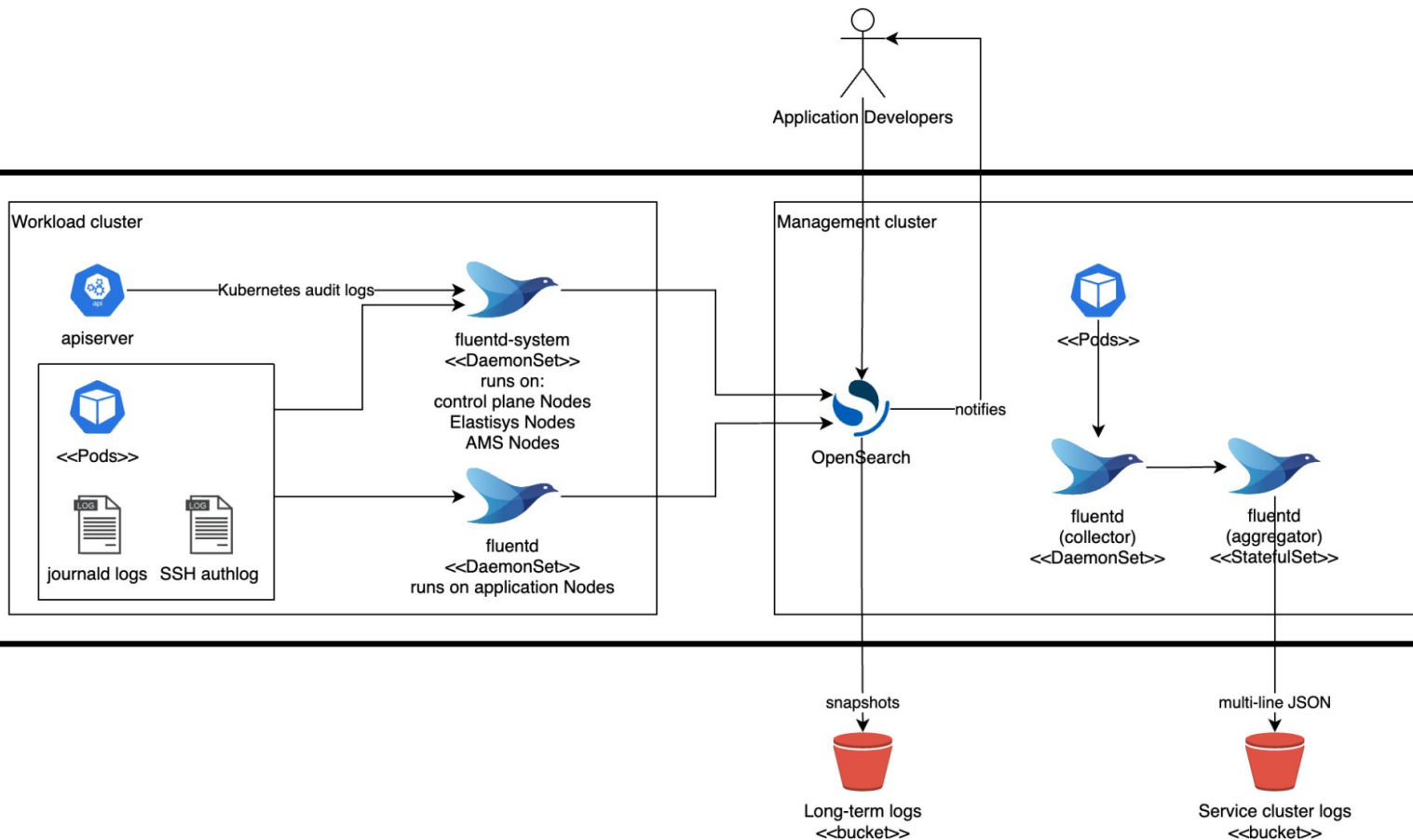
- ✓ Log exceptions and errors
- ✓ Structured logs in multi-line JSON format on stdout
- ✓ Log levels
- ✓ Implement app specific dashboards and/or alerts



Customer



Infrastructure layer





Automation

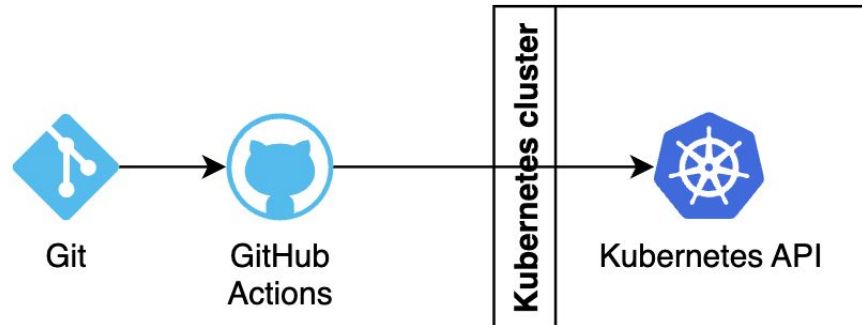
Provided by Platform

- ✓ CI Solution
- ✓ Container Registry
- ✓ Vulnerability Scanning
- ✓ CD Solution
- ✓ Golden Path

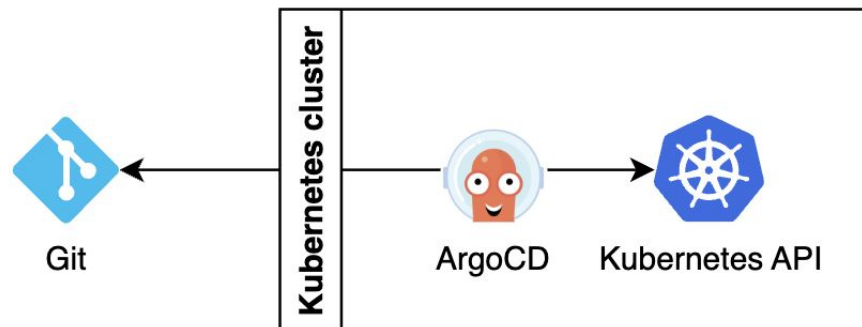
Implemented by App Devs

- ✓ App Specific Automated Testing
- ✓ Utilizing Golden Paths

Push-style: Changes are pushed from Git to the Kubernetes cluster



Pull-style: Changes are pulled by the Kubernetes cluster from Git





Security by Design

Provided by Platform

- ✓ **Security-hardened** Kubernetes
- ✓ **Security by Default** settings

Implemented by App Devs

- ✓ **Application-level** security
- ✓ **Separate secret** from non-secret configuration information



Minimal Need to Know

Provided by Platform

- ✓ Configure **namespaces** to **enforce Restricted** Pod Security Standard
- ✓ **Prevent** from running as **privileged user**

Implemented by App Devs

- ✓ Ensure that app **doesn't require** elevated **privileges**
- ✓ Create container images that run as **non-root**



Defence in Depth

Provided by Platform

- ✓ **Prevent** running containers with **vulnerabilities**
- ✓ **Require** specification of resource **quotas**
- ✓ **Require** specification of **Network Policies**

Implemented by App Devs

- ✓ Keep your **dependencies up to date**
- ✓ **Specify** resource **quotas**
- ✓ **Implement** fine-grained **Network Policies**



Resilience

Provided by Platform

- ✓ High-Availability
Kubernetes setup
- ✓ Support probes
 - ✓ Load balancing
 - ✓ Restarting container
 - ✓ Rolling updates

Implemented by App Devs

- ✓ Implement Startup, Readiness and Liveliness probes
- ✓ Fail startup probe during DB migration
- ✓ Fail readiness probe if downstream service is unavailable
- ✓ Handle SIGTERM by failing readiness probe



Overview

Mission and Vision

For Application Developers ▾

Overview

Understand the Basics

Getting Started >

Go Deeper ▾

Prepare Your Application

Prepare Your Identity
Provider (IdP)

Container registry

Continuous Development

GPU

Network Model

Kubernetes API

Kubernetes UI

Namespaces

Logs

Log-based Alerts

Long-term Log Retention

Metrics

Metric Alerts

Maintenance

Backups

List of Non-Functional Requirements ¶

In some contexts, it is useful to have a more-or-less exhaustive list of **non-functional requirements** ↗ which the application needs to fulfill to make the best use of the underlying platform. Sometimes, these may loosely be called "IT requirements".

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119](#) ↗.

Code	Requirement	Justification for exclusion OR Evidence OR Comment
R1	Architectural requirements	
R1.1	Application components MUST be loosely coupled. Data sharing MUST happen exclusively via versioned APIs.	Why is this important? >
R1.2	The application MUST use separate containers for synchronous API requests and running asynchronous batch jobs.	Why is this important? >

Table of contents

Make Sure Your Application Can
Terminate Gracefully

Make Sure Your Application
Tolerates Nodes Replacement

Further reading

[List of Non-Functional
Requirements](#)





Key Takeaways



Cloud Native Application Platforms help you with fulfilling NFRs.



There is still a lot of work to be done by App Devs on the app level.

Thank You!



Jakub Krzywda

<https://www.linkedin.com/in/jakubkrzywda/>

jakub@elastisys.com