Xebia

Journey to Cloud Native

Our views and experiences on cloud transformation

Agenda

Introduction

The cloud journey

Begin with a solid foundation

Lessons learned / Tips

Why I love Cloud

Working with tens of companies in just three years

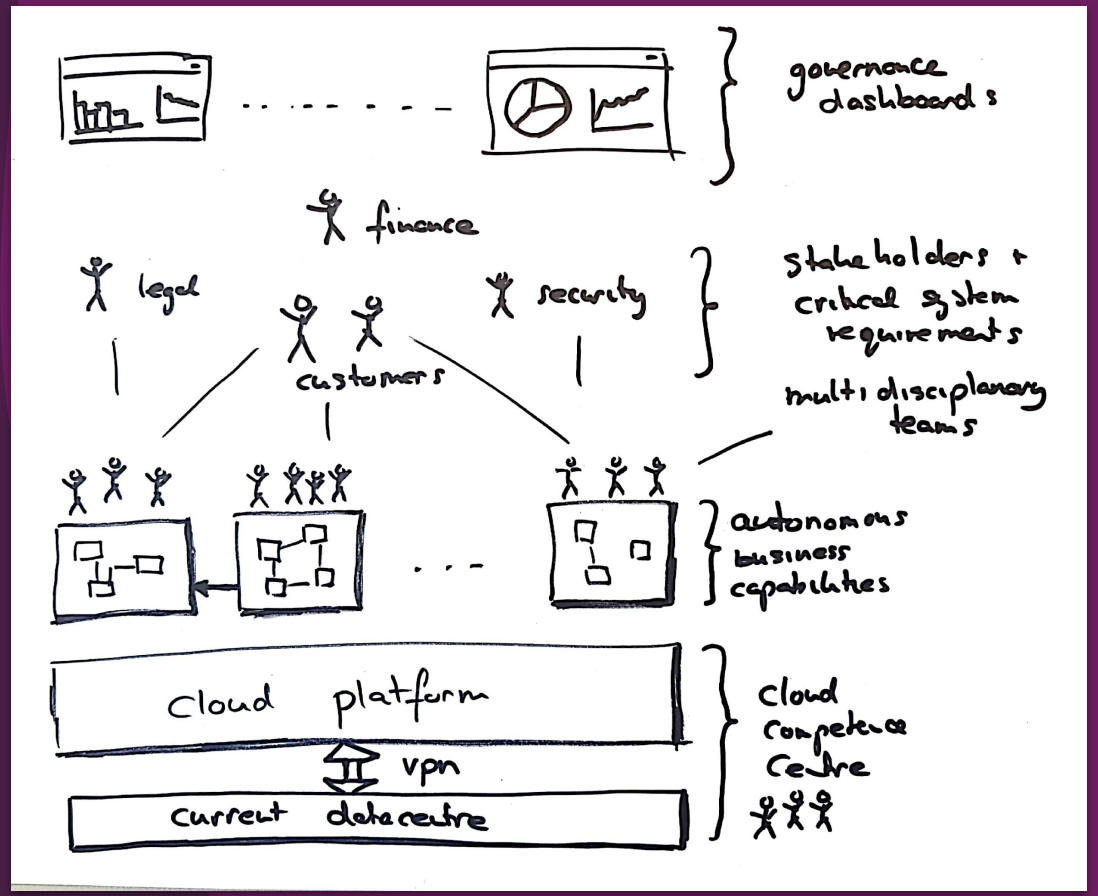

Defining
Due North

Defining Cloud-Native

“Cloud-native” is an approach to building and running applications that exploits the advantages of the cloud computing delivery model. “Cloud-native” is about how applications are created and deployed, ***not where****.*

Cloud-native transformation

- ➔ Customer-Centric Action
- ➔ Create with the End in Mind
- ➔ End-To-End Responsibility
- ➔ Cross-Functional Autonomous Teams
- ➔ Continuous Improvement
- ➔ Automate Everything You Can



Cloud Native Migration Principles

- ➔ **Simplicity**
- ➔ **Autonomy**
- ➔ **Design for failure**
- ➔ **Minimize waste**
- ➔ **Minimize manual work**
- ➔ **You build it, you run it**
- ➔ **SaaS over build**
- ➔ **Open platform**

- ➔ **Infrastructure as code**
- ➔ **Everything is under version control**
- ➔ **Immutable Infrastructure**

- ➔ **Continuous Delivery Build Pipelines**
- ➔ **Automated System Test**
- ➔ **Continuous Business Service Monitoring**
- ➔ **Active-Active High Availability**
- ➔ **Self-Healing**

Cloud-native Platform Principles

- ➔ **Simplicity** - ”*Simplicity is a prerequisite for reliability – Edsger Dijkstra*” Keep the architecture and solutions as simple as possible. It ensures the systems are robust, easy to secure and easy to maintain.
- ➔ **Design for failure, experience none** - We understand that the world is imperfect. Designing for failure means that we can detect errors, provide graceful degradation of the service and automatically recover from errors.
- ➔ **Minimize Waste** - Minimize waste both in processes and systems. Waste is everything that does not contribute to customer value.
- ➔ **Minimize manual work** - A special form of waste reduction: manual work introduces delays and increases complexity and room for error.
- ➔ **Autonomy** - Ensure that both systems and teams can operate in an autonomous fashion.
- ➔ **You build it, you run it** - There is one team responsible for the development, continuous improvement, reliability and availability of the business capability in production.
- ➔ **SaaS over Build** - Whenever required functionality is available as a service from the Cloud Provider, it is prefered over building your own.
- ➔ **Open platform** - Whenever an open platform solution is available, it is preferred over a vendor specific solution. You want to keep your options open.

Cloud-native Configuration Management Principles

- ➔ **Everything is under version control** - All changes to the software and the infrastructure are under version control. Manual changes in deployed software and environments are strictly prohibited.
- ➔ **Infrastructure as code** - The entire infrastructure of the IT landscape is programmed as code. Infrastructure as code allows the infrastructure to be version controlled, traceable and testable. In addition, once your entire landscape is defined as code, disaster recovery is built-in. Recreating your entire landscape in another geographical region of the same Cloud provider is a standard feature.
- ➔ **Immutable Infrastructure** - The entire infrastructure is specified as code, automatically built and deployed. Manual changes in deployed software and environments are strictly prohibited and even made impossible. All applications are delivered and built as Docker Images or Machine images.

Cloud-native quality assurance principles

- ➔ **Continuous Delivery Build Pipelines** - Both the infrastructure and the applications are deployed through continuous delivery pipelines. The entire software delivery process from source code to production is automated.
- ➔ **Executable specifications** - The desired behaviour of the software services is specified as executable specifications. These are included in the software delivery pipeline and automatically evaluated.
- ➔ **Automated System Test** - All services have automated system tests which are included in the continuous delivery pipeline, preventing defects from reaching production.
- ➔ **Continuous Business Service Monitoring** - The functional availability of the business services is monitored 24/7.
- ➔ **Active-Active High Availability** - Both the IT infrastructure and the applications are deployed across 2 or more availability zones (e.g. data centres in a region), providing protection against single data center failures.
- ➔ **Self-Healing** - The Infrastructure and the applications are self-healing and capable of solving common error situations (single application crashes, machine crashes, data centre loss, etc.).

Obvious cloud-native goals

➔ Increase speed of value delivery

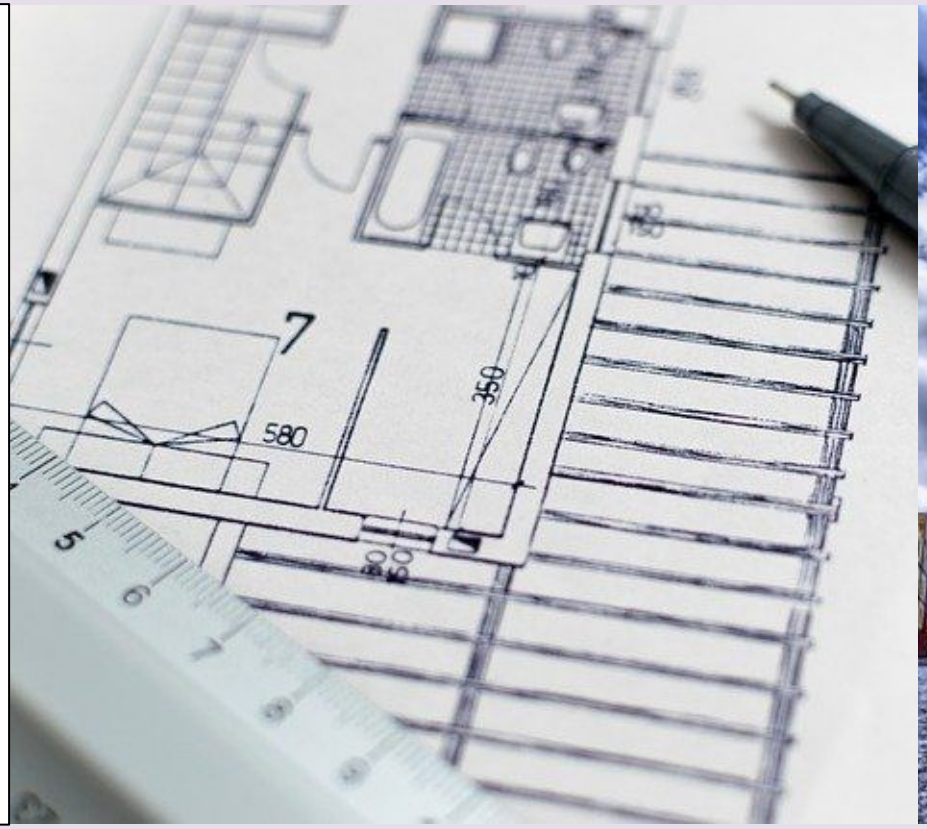
➔ Increase predictability

➔ Decrease operational effort (No/LowOps)

➔ Effortless scaling

➔ Changing cost model

Obvious ways to move to the cloud



Xebia

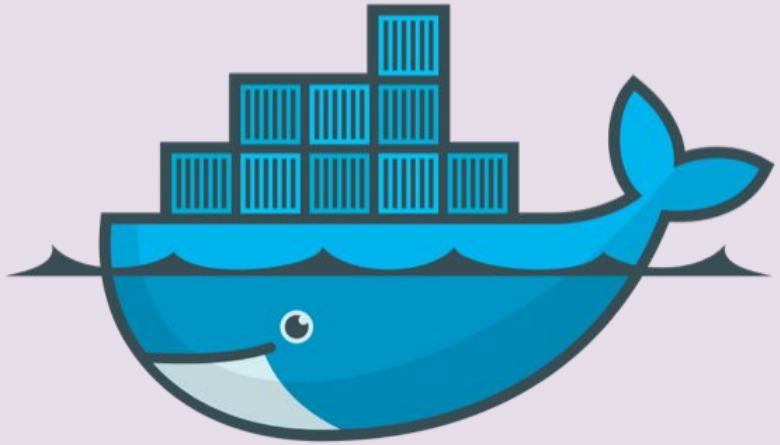
What if your applications are too vast to redesign?

Cloud-native roadmap

- ➔ Containerize applications
- ➔ Utilize cloud services
- ➔ Standardized infrastructure
- ➔ Infrastructure as code
- ➔ Continuous delivery pipelines
- ➔ Measure Everything

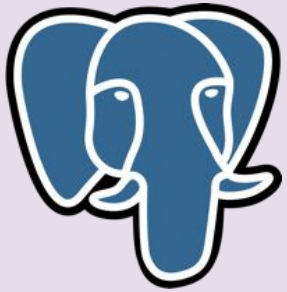
Containerize Applications

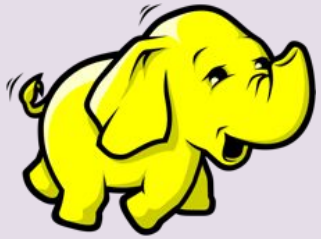
- ➔ Basis for platform standardization
- ➔ Runs everywhere
- ➔ 100% identical
- ➔ Local system tests

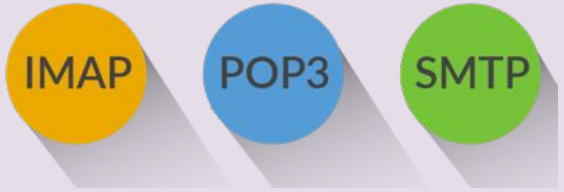


Utilize Cloud Services

- ➔ Ease of Use
- ➔ Low Maintenance
- ➔ High Availability
- ➔ High Scalability
- ➔ Pay For Use







Standardized Infrastructure

➔ De-facto standard for Container based workloads

➔ Declarative application deployments

➔ Runs on any Cloud Provider

Infrastructure as Code

➔ Ensures all environments are identical

➔ Manual changes to infrastructure are forbidden

➔ Infrastructure changes are tested too

Continuous Delivery Pipelines

- ➔ All code changes are stored under version control
- ➔ Committed changed lead to new versions of applications and infrastructure configurations
- ➔ Automated tests are run on every commit

Measure Everything

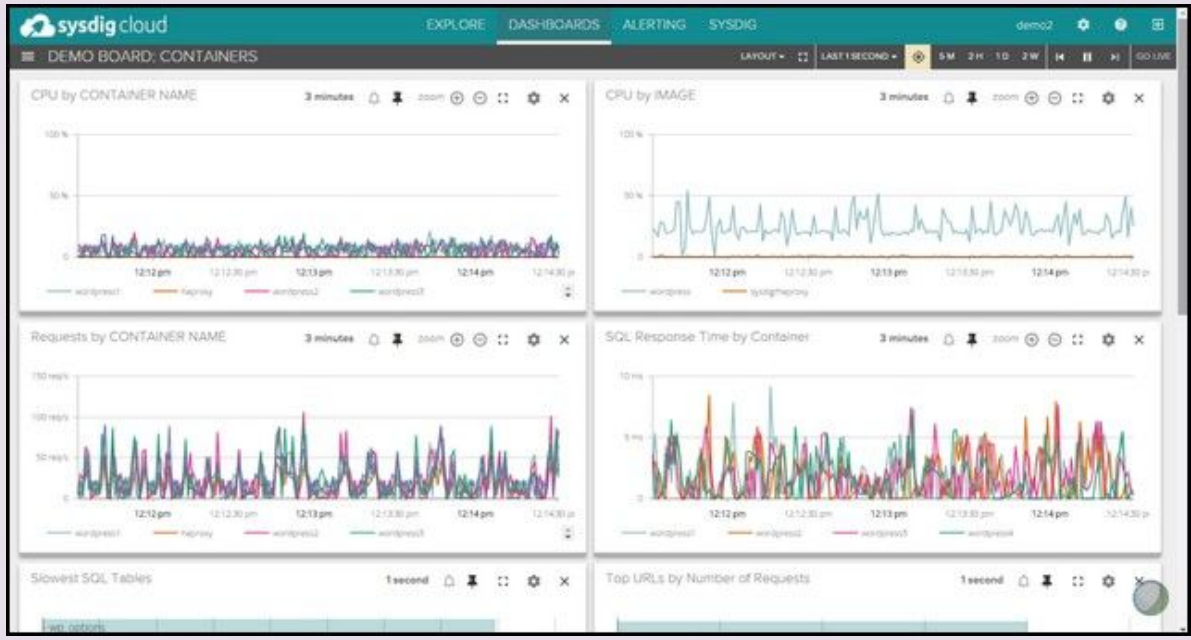
➔ Focus on Service Level Indicators

Basis for:

➔ Continuous Improvement

➔ Incident response

➔ Post mortems



Isolate your Workloads and/or Environments

- ➔ Cost overview per ...
- ➔ Reduced blast radius

Assume a role to play!

Users can pick a role to assume in an account. The role determines what they can do:

- Developer
- Operator
- Product Owner

Networking

How many application developers do you know that understand networking?

Security and Governance

- Depending on the industry
  - Open vs Closed
- Centralised tooling
  - Security Findings
  - Compliance status
  - Audit Logs

Other shared services

- On-premise connectivity
- Golden AMIs
- etc

Landing Zone

Includes all the automation needed to get a workload started as soon as possible.

It acts as a boilerplate for your projects, and it will:

- ➔ Reduces the time to market.
- ➔ Stimulates experimentation.
- ➔ Removes complexity.

Lessons learned

Lessons Learned / Tips

➔ Setting up a full blown landing zone takes time

➔ Start small, iterate and improve over time

➔ Focus on what you really need

➔ Automate repetitive tasks as soon as possible

➔ Goal is the workload, not the landing zone

Questions?

Joris Conijn - https://www.linkedin.com/in/jorisconijn/
Laurens Knoll - https://www.linkedin.com/in/laurensknoll/