OpenShift @ SBB

Java User Group – 27.9 & 25.10.17

Tobias Denzler, Philipp Oser

Who we are

Tobias Denzler

- Software Engineer at SBB IT
- Java & OpenShift enthusiast
- @tobiasdenzler

Philipp Oser

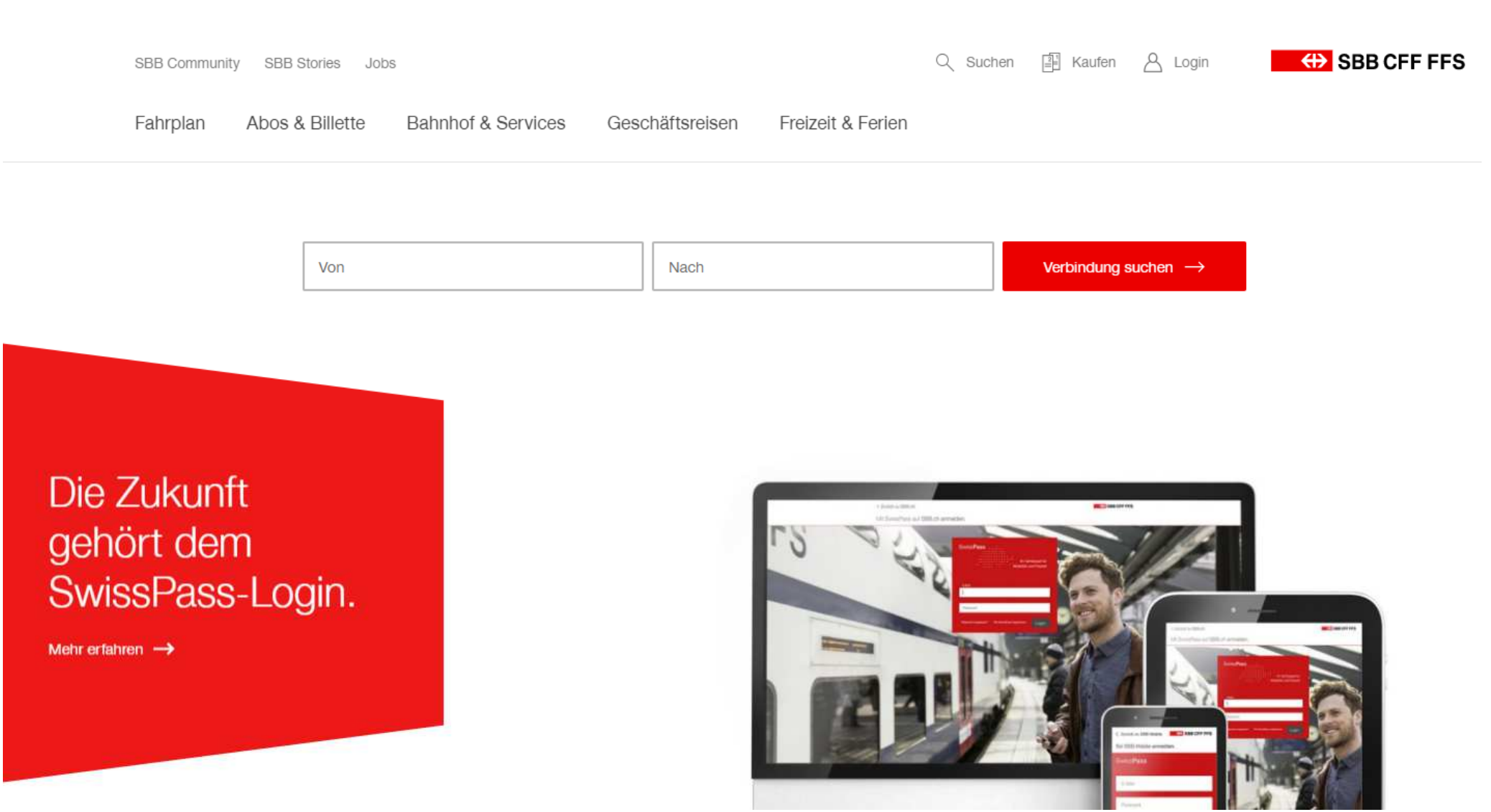
- Architect at ELCA
- Interest in «good abstractions» for devs
- @silsfan

Agenda

1. Intro
2. Demo
3. More advanced topics
4. Conclusion
5. Q&A

30k requests per minute

Mobile app backend



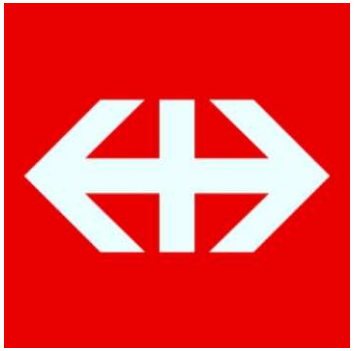
www.sbb.ch – our main website

25k requests per minute

Monitors at all stations

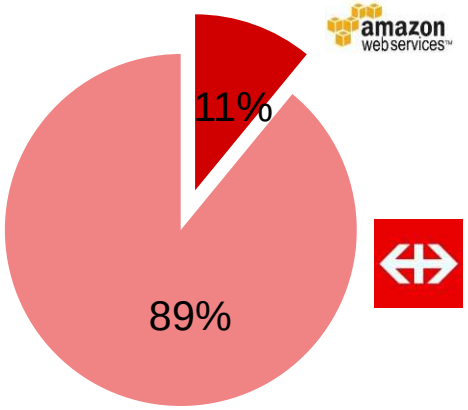
Numbers

OpenShift Clusters

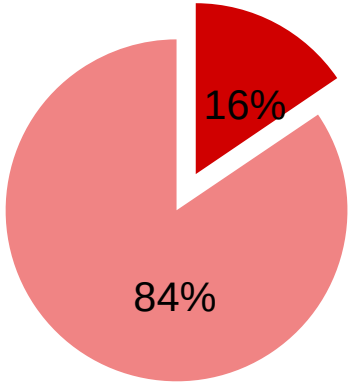


Master nodes	2	2
Storage nodes	2 > 6	2 > 6
Working nodes	26	7
Compute	1250 Cores	56 Cores
Memory	13 TB	400 GB
Persistent Storage	40 TB	200 GB
Datacenter	2 sites	2 availability zones

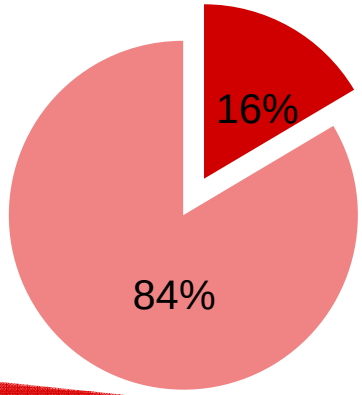
2700 Containers



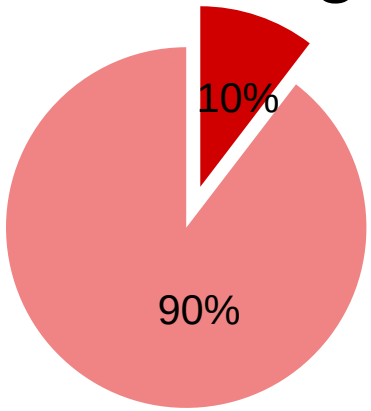
970 Projects



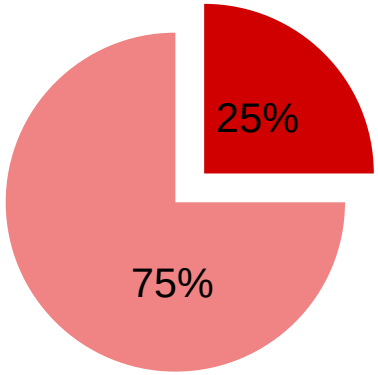
900 Applications



6500 Images



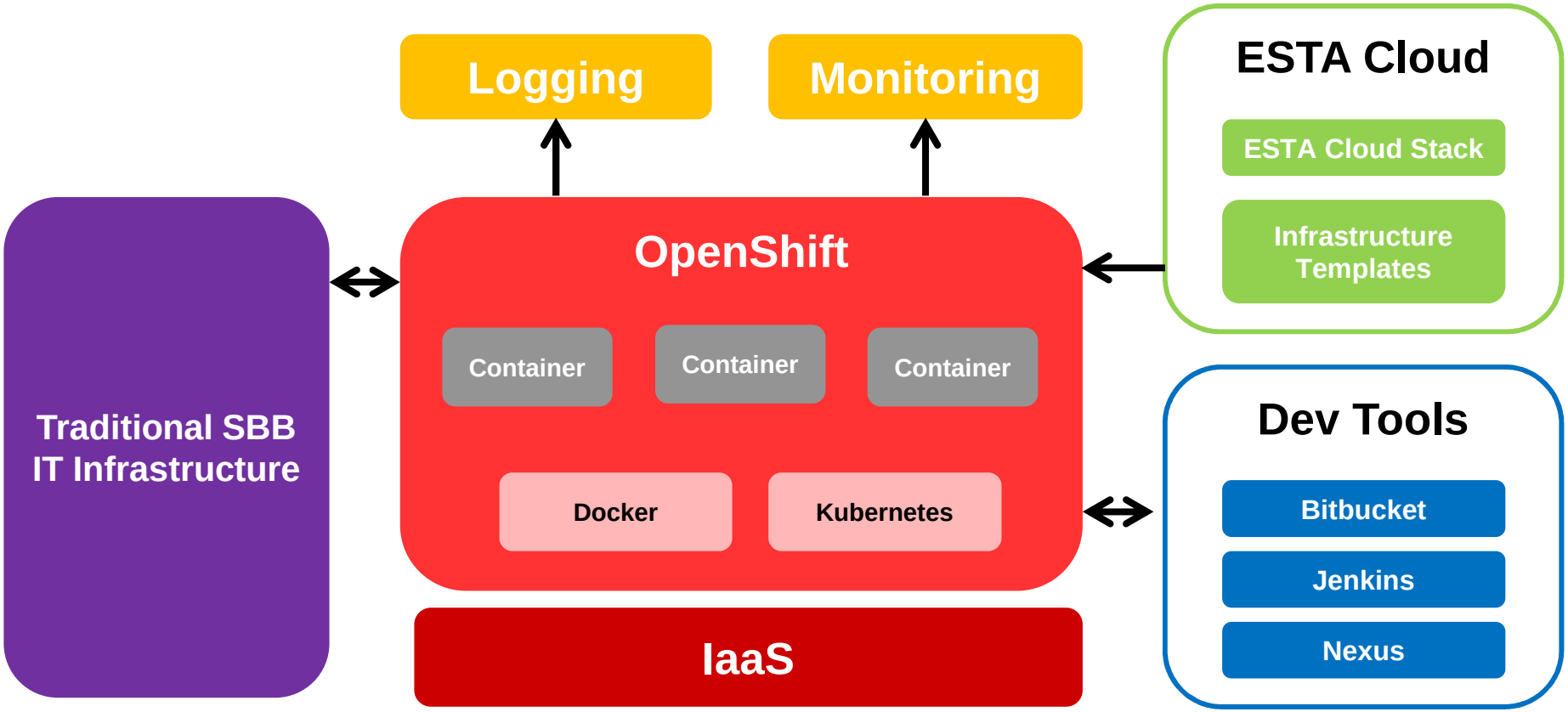
50 Mio Requests/ Day



Numbers

PaaS

Context

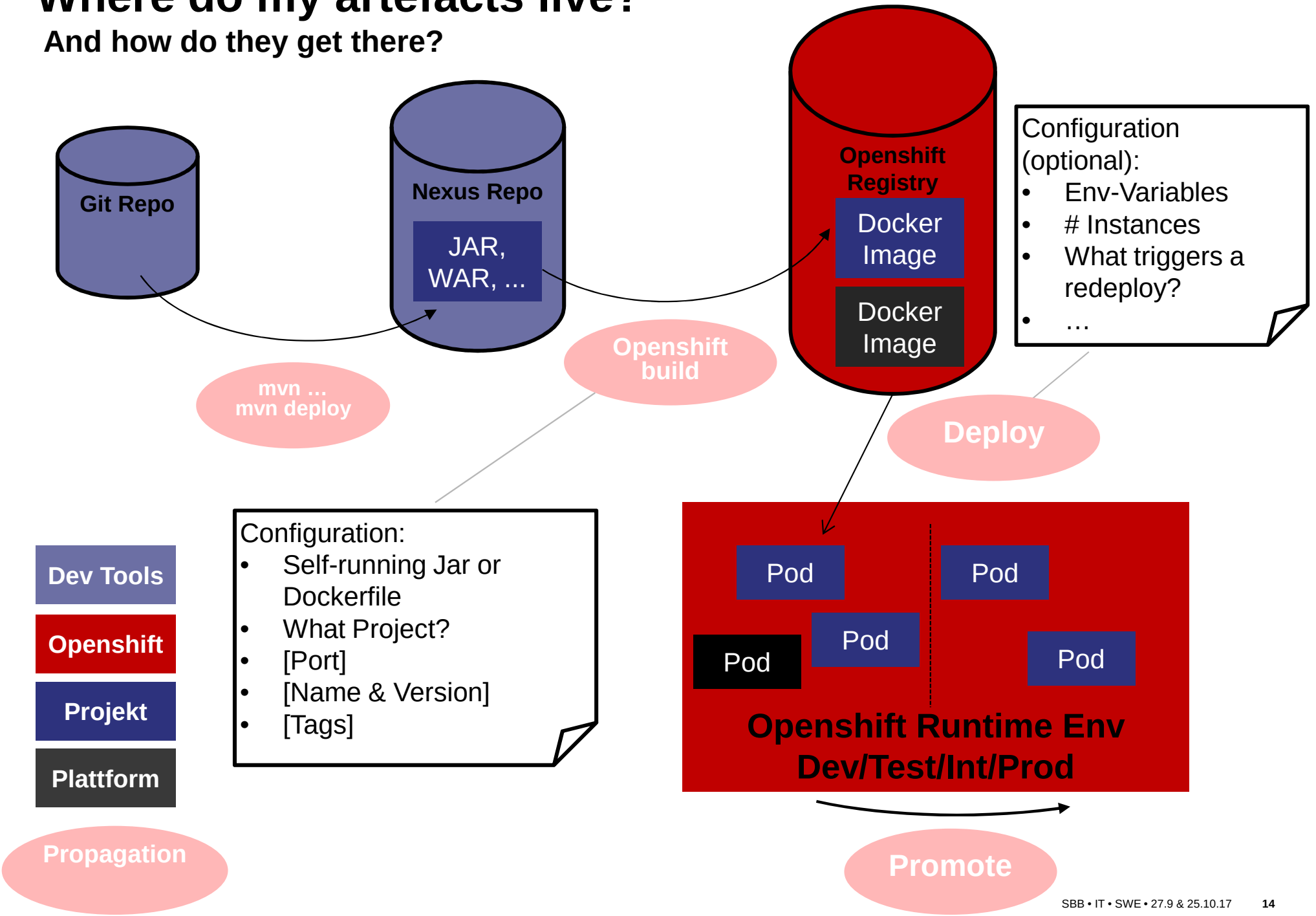

Logging
Monitoring
ESTA Cloud
ESTA Cloud Stack
Infrastructure Templates
OpenShift
Container
Container
Container
Traditional SBB IT Infrastructure
Docker
Kubernetes
Dev Tools
Bitbucket
Jenkins
Nexus
IaaS

Layering

Demo

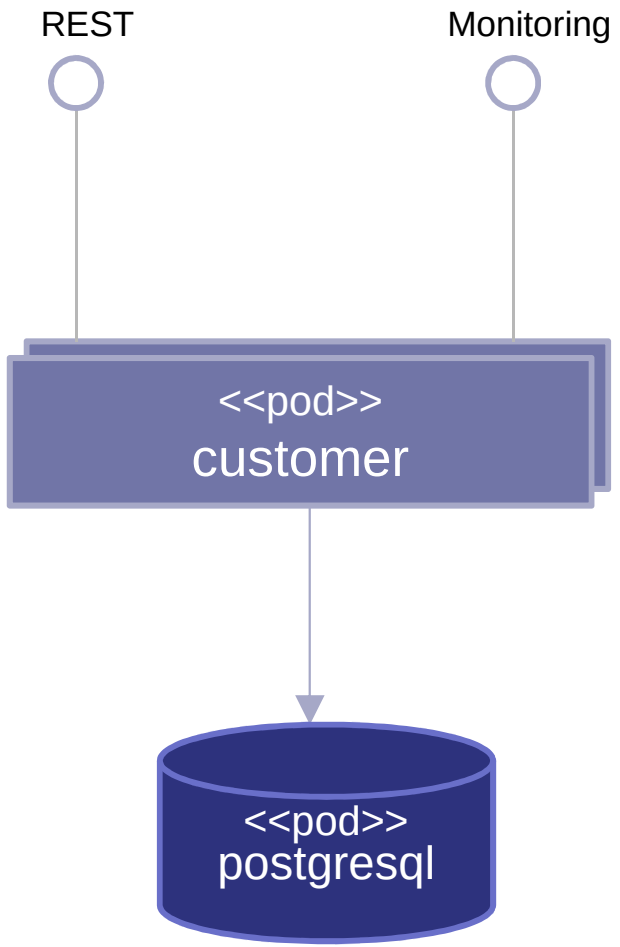
Where do my artefacts live?

And how do they get there?



Demo-Steps

- Initializr
- Jenkins Pipeline
- Openshift UI
- Rolling Deployment
- Monitoring
- Reliability



More advanced topics

Testing

- Good practices remain:
  - Unit-, Integration Tests, …
  - Propagation through Stages (dev, test, …, prod)

- Resilience Testing
  - Chaos Monkey
  - Toxi-proxy Injection via script
  - Remote Test Runner

Other topics (selection)

- Templates & Images: Postgresql, Elasticsearch, Cassandra, Hazelcast, RabbitMQ, NGINX, Java Base Image …
- Configuration of «desired state» in Openshift – over N stages, automatic, reproducible, management of changes
  - Desired state in Git
- Graceful Shutdown for Spring and Openshift
- Checklist for productive deployments
  - naming, > 2 pods, limit resources, monitoring, crash resilient
- Migration to the Cloud: e.g. Only expose HTTP protocols, Sessions
- Open source components for Openshift: https://github.com/oscp

Conclusion

Traditional Infrastructure vs PaaS

	Traditionell	PaaS
Technology	Shared Websphere Platform	Openshift Cluster
What deployables can run?	Websphere Java EE Deployables (2 fixed Versions & Patch-Levels)	Docker Container
Deployment to Prod	Description in Word Document (!)	Self-service Automatic (or via UI)
Cost		Lower (about - 30%)
System Know-How	Specialized Team for technology	DevOps-Team (Cloud Team 2nd Level)
Scalability/ New Environment	Static (weeks)	Dynamic (seconds)
Governance	Technology limited Architecture review	Governance via «what is easy»

Cool

- Surprised by huge success at SBB!
- Self-service IT, very flexible & relatively simple «close to ideal infrastructure»
- Templates: Predefined components, can be instantiated via self-service mode, automatically supervised

Challenges

- Templates: Large Effort for Design, Maintenance & Know-How
- Know-How in DevOps Teams: Huge technology stack (including e.g. details of Linux distros, Docker, JVM, monitoring, …)
- Full Self-Service ability: Tools supporting multi-tenancy, own mini-Uis, Security, vs flexibility

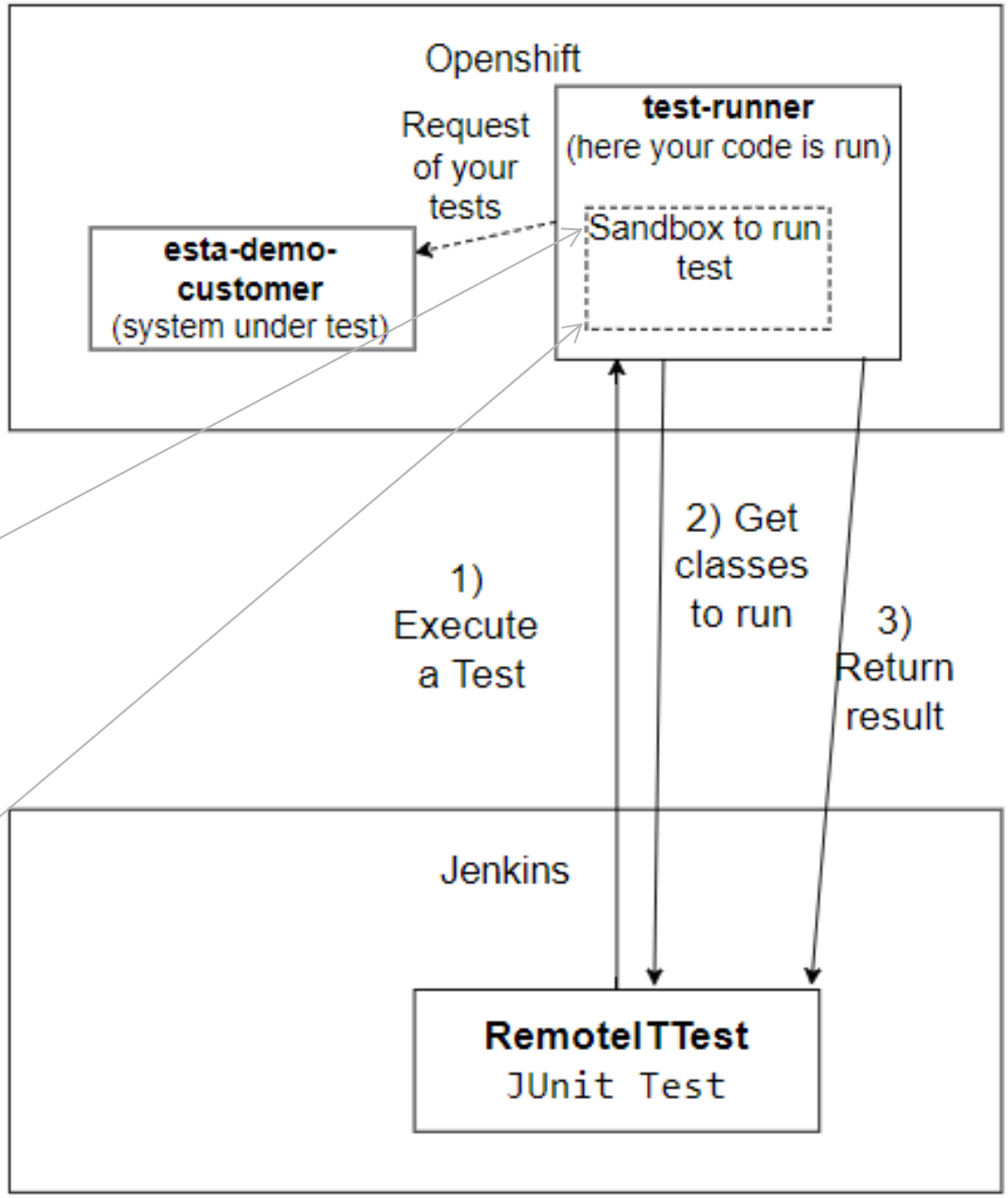
Questions?

Test runs on Openshift, driven from Jenkins

- Convenient pgm model
- Internal Ports accessible

```
@RunWith(RemoteTestRunner.class)
public class SimpleJUnitTest {

  @Test
  public void testRunInOpenshift() {
     x = codeToRunInOpenshift();
     assertEquals(x, "foo");
  }
}
```



Build & Deployment

- → how to manage "desired state" of openshift project
  - 2 philosophies
    - syncher pod (desired state in git)
    - script to update desired state (e.g. launched in jenkins)
  - uses templating (for different envs)
  - Promotion to next env is then via script/ git update