GDC CLOUD SCHULUNG
PRACTICE DEVOPS

Bernd Rederlechner, 26.10.2017

T · · Systems ·

AGENDA

01	**Intro: Continuous Delivery Pipeline revisited**
02	Reproducibility: Everything as Code
	02.1 Exercise: Trigger pipeline (Jenkins from GIT)
03	Provisioning time: Infrastructure as Code
	03.1 Exercise: Automated install and configuration
04	Dev-Prod-parity: Immutable Infrastructures
	04.1 Exercise/Video: Image bakery
05	Conways law and continuous feedback
	05.1 Exercise: ELK logging as microservices
06	Learn: Red/black, A/B and canaries
	06.1 Exercise: My little A/B test

PATH 1:
Optimize flow

PATH 2:
Include feedback

PATH 3:
Learn from system

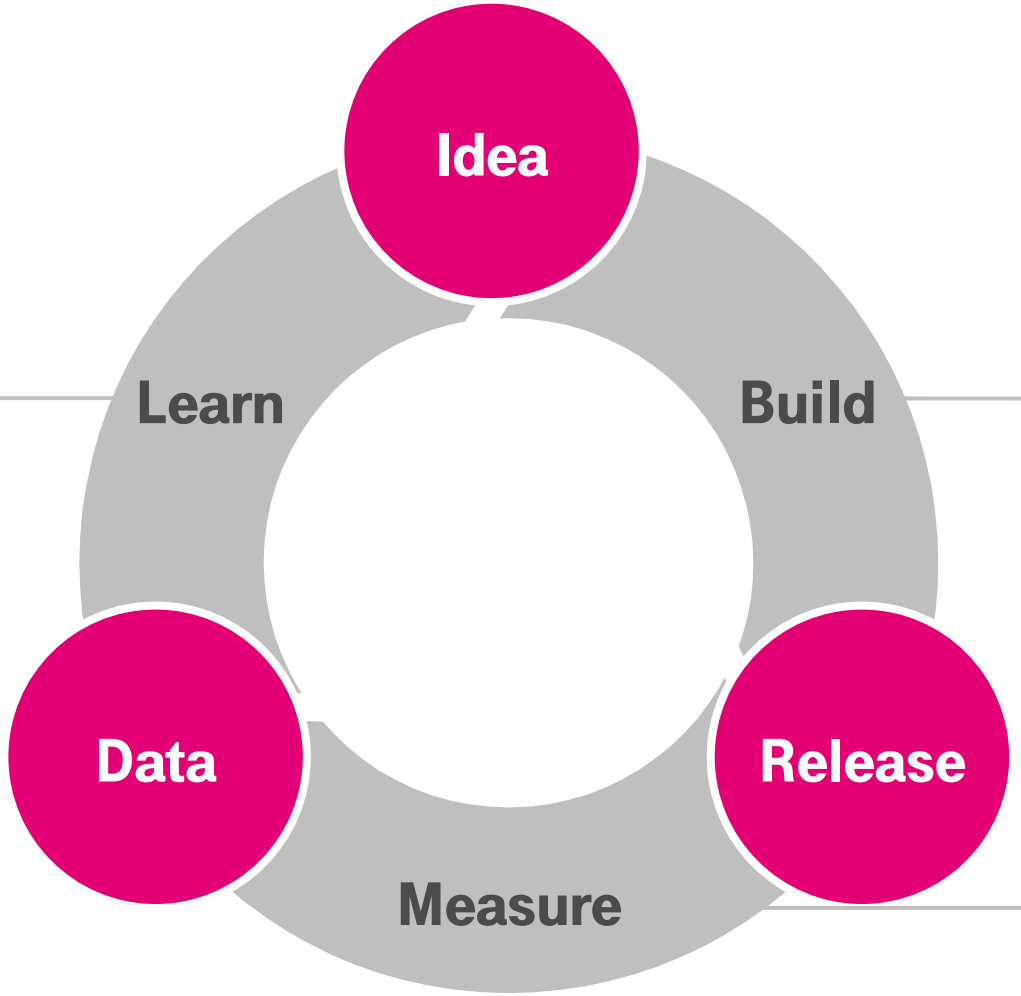
T · · Systems ·

HINWEIS ZUM COPYRIGHT

Diese Schulungsunterlage ist ausschließlich für die im Titel genannte Veranstaltung zu verwenden. Eine Weitergabe der gesamten Präsentation oder einzelner Folien sowie eine Verwendung der Inhalte in anderen Vorträgen ist nicht gestattet.

1 CONTINUOUS DELIVERY REVISITED

CONTINUOUS IMPROVEMENT - KAIZEN



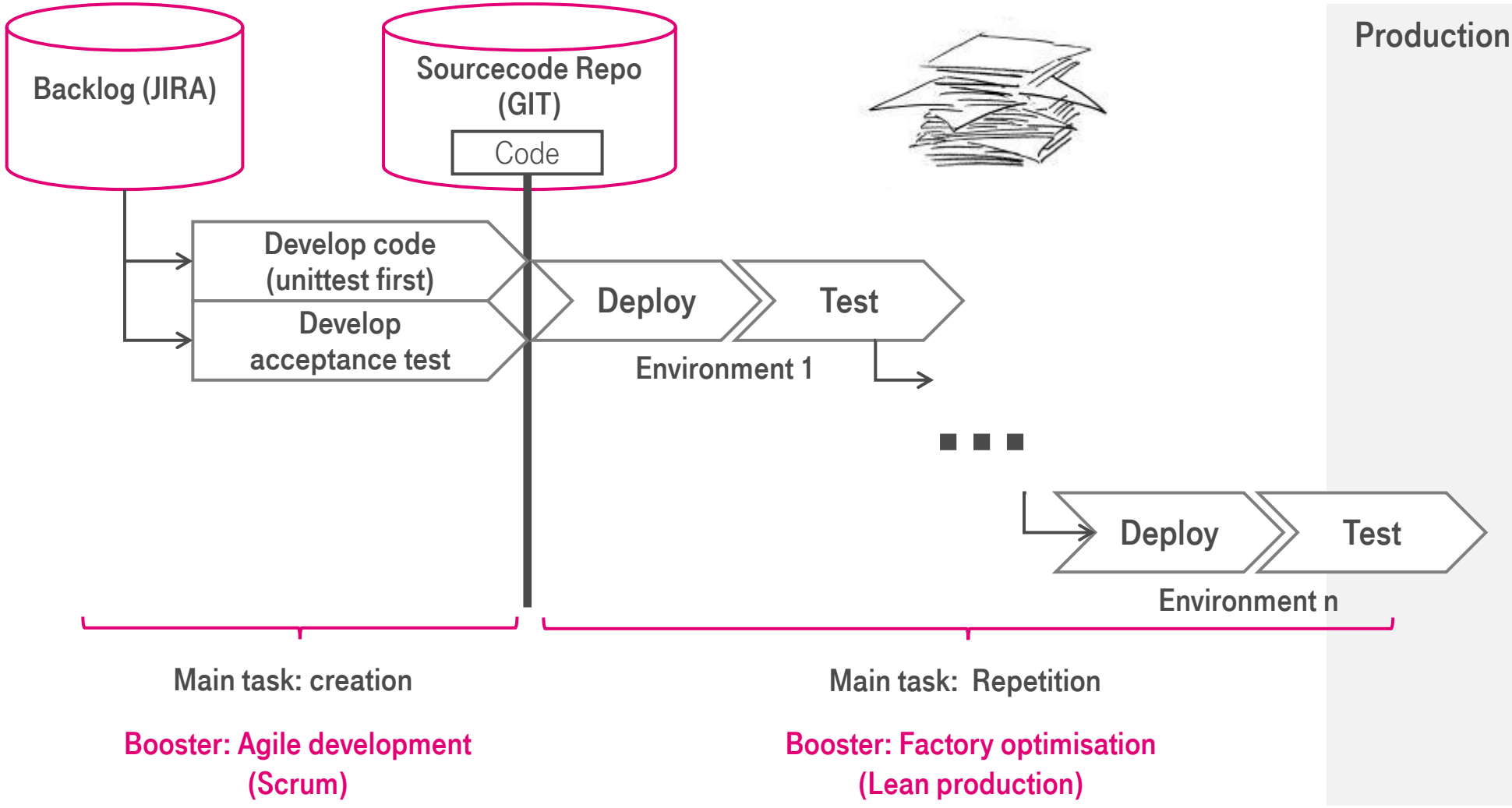
Path 3
Learn with your customer

Path 1
Optimize release flow

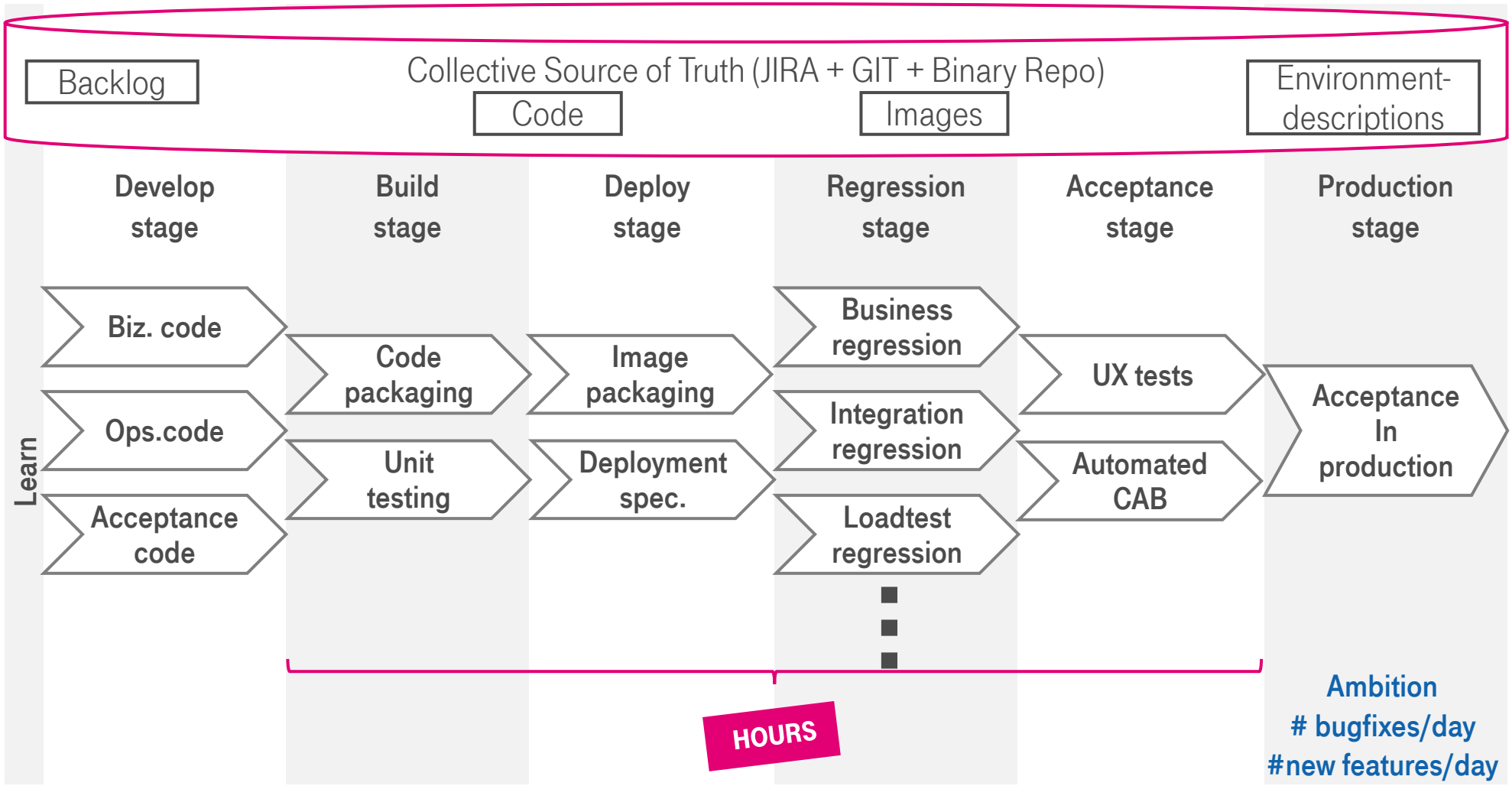
Path 2
Include feedback

T · · Systems ·

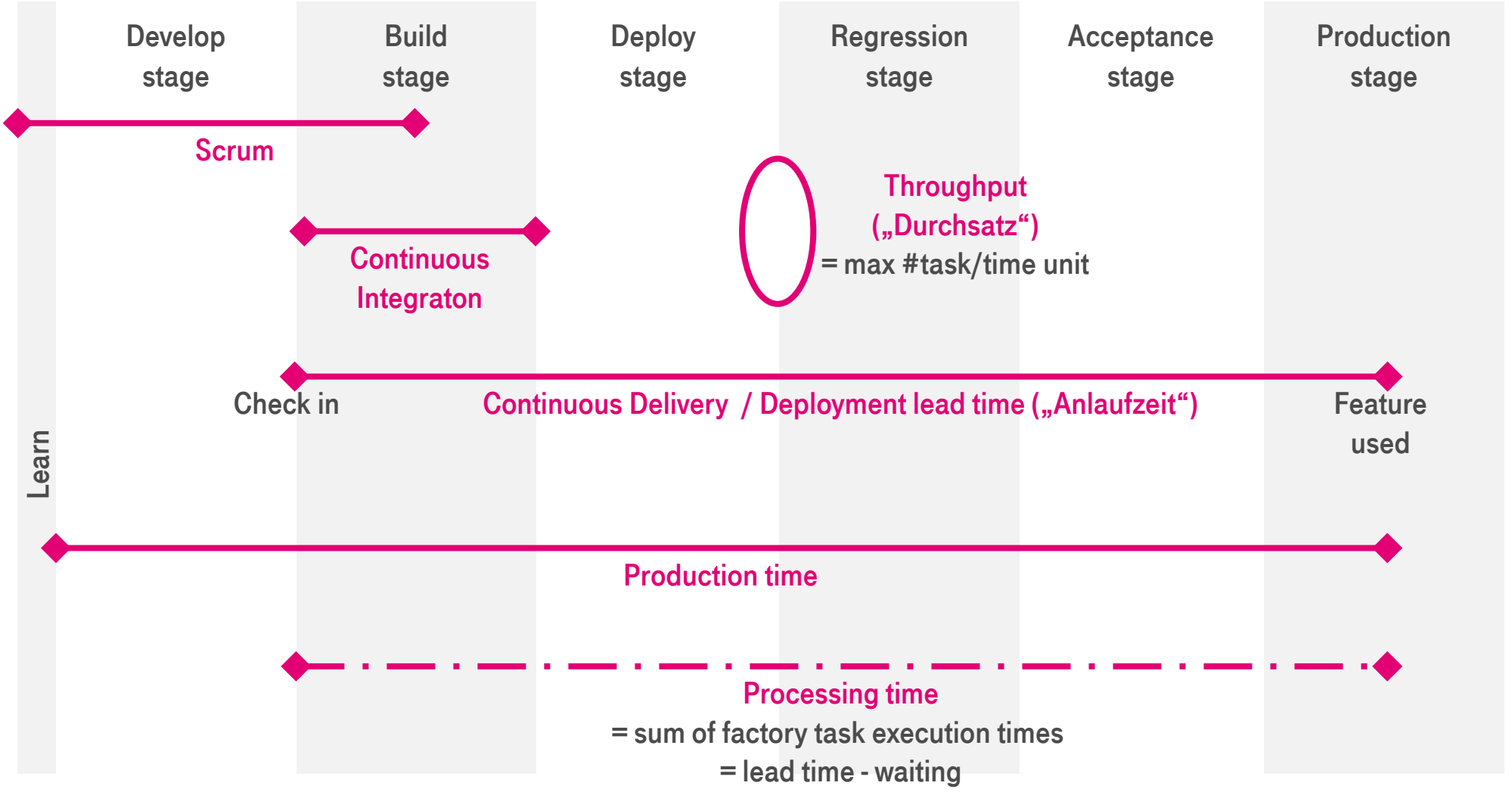
END-2-END AGILE SOFTWARE PRODUCTION
AS-IS PROCESS VIEW



END-2-END AGILE SOFTWARE PRODUCTION
TO-BE PROCESS VIEW



WORD DEFINITIONS AND TIME INTERVALS



THE SEVEN WASTES OF SOFTWARE DEVELOPMENT

#1 - Partially Done Work (Inventory)

#2 - Extra Features, unused features (Overproduction)

#3 – Relearning (Extra processing)

#4 – Handoffs (Transportation)

#5 – Delays (Waiting)

#6 - Task Switching (Motion, Jumping)

#7 - Defects

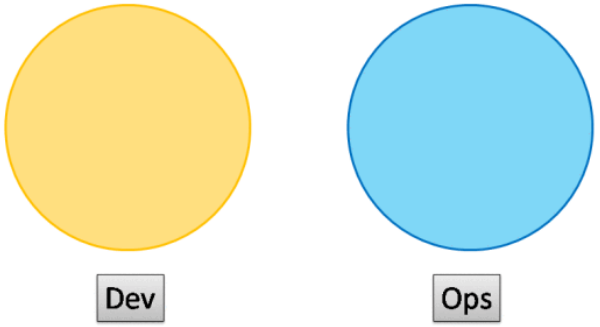
> You build it, you run it.
>
> **Werner Vogel, Amazon**

T · · Systems ·

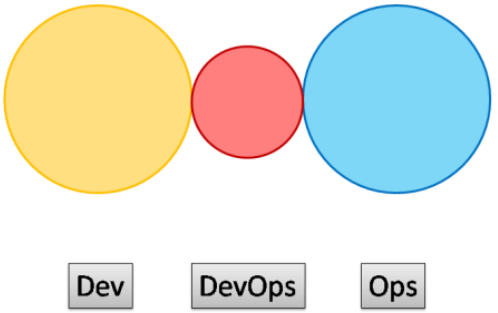
HANDOFFS: EXPERT TEAMS REPLACED BY CROSS-FUNTIONAL TEAMS

TEAM-STRUCTURES: DEVOPS ANTI-TYPES

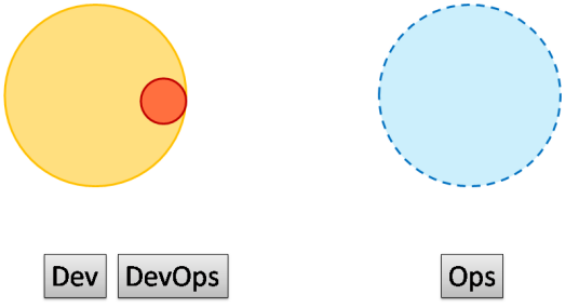
Anti-Type A – Separate Silos



Anti-Type B – Separate DevOps Silo



Anti-Type C – "We Don't Need Ops"



DevOps Patterns:
Team Topologies

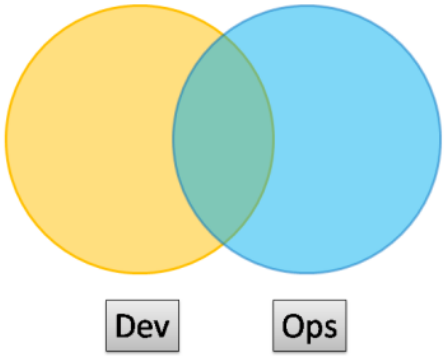
Matthew Skelton
@matthewpskelton
softwareoperability.com + experiencedevops.org

T · · Systems ·

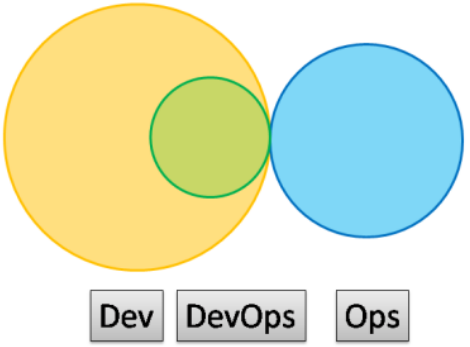
TEAM-STRUCTURES: DEVOPS POSITIVE PATTERNS

REMEMBER: CROSS-FUNTIONAL IS MANDATORY

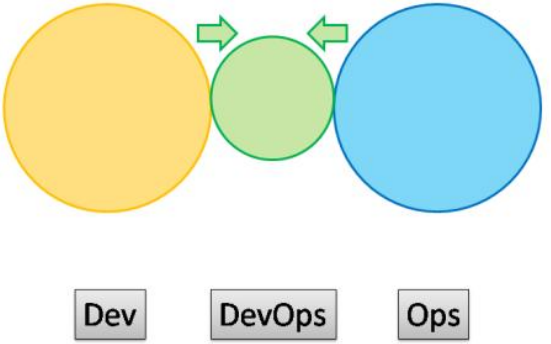
Type 1 – Smooth Collaboration



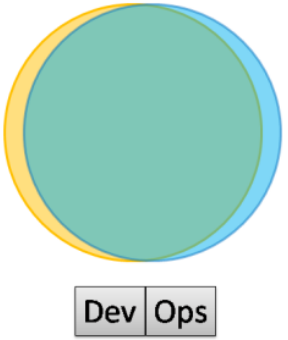
Type 3 – Infrastructure-as-a-Service



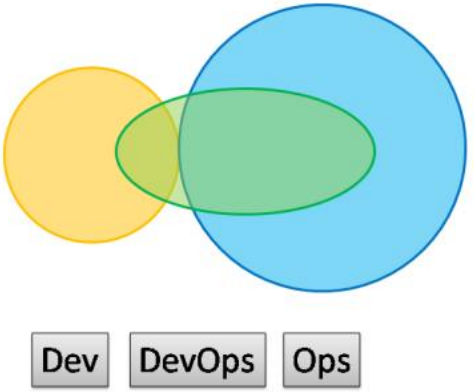
Type 5 – Temporary DevOps Team



Type 2 – Fully Embedded



Type 4 – DevOps-as-a-Service



T · · Systems ·


KAFFEE-
PAUSE

AGENDA

01	Intro: Continuous Delivery Pipeline revisited
02	Reproducibility: Everything as Code
	02.1 Exercise: Trigger pipeline (Jenkins from GIT)
03	Provisioning time: Infrastructure as Code
	03.1 Exercise: Automated install and configuration
04	Dev-Prod-parity: Immutable Infrastructures
	04.1 Exercise/Video: Image bakery
05	Conways law and continuous feedback
	05.1 Exercise: ELK logging as microservices
06	Learn: Red/black, A/B and canaries
	06.1 Exercise: My little A/B test

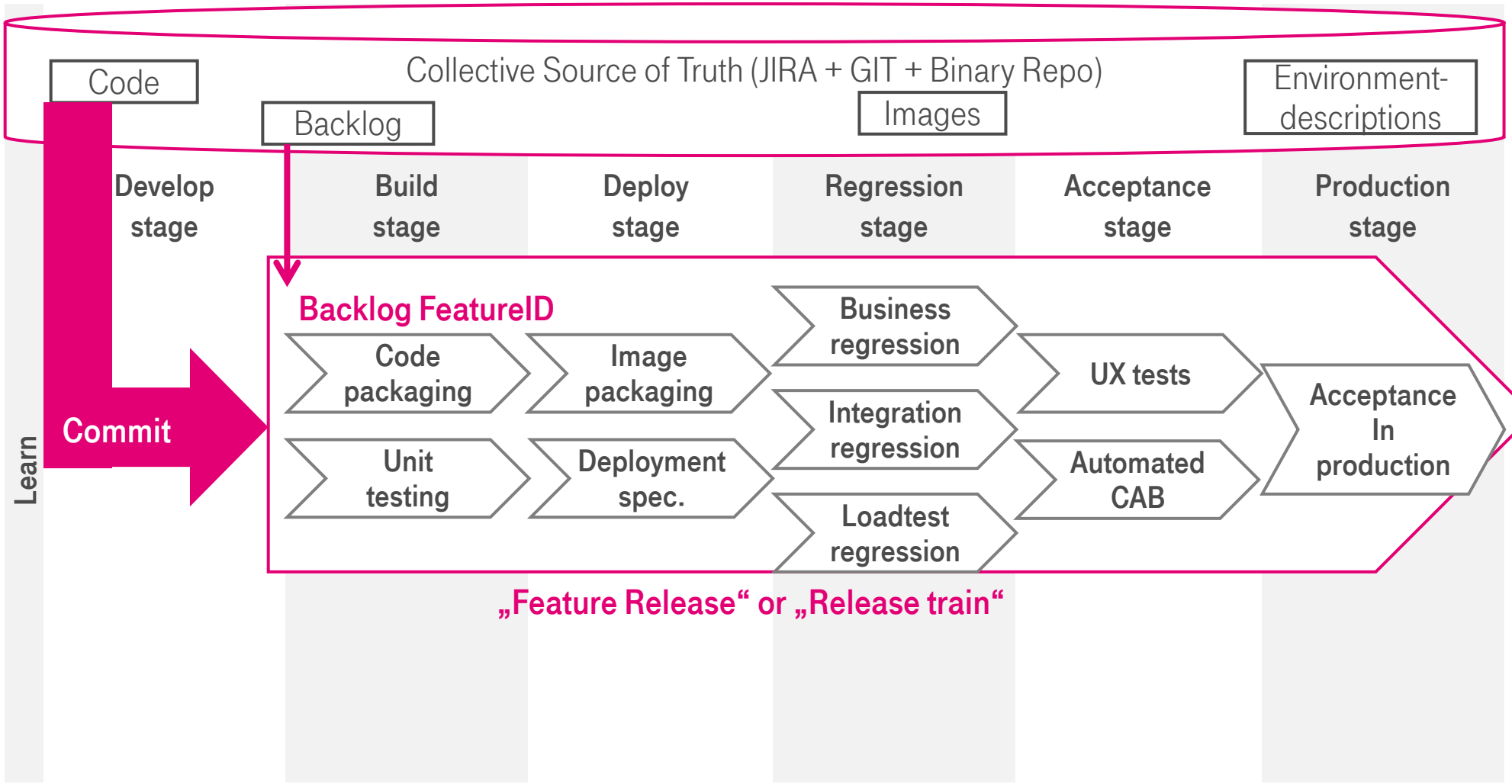
PATH 1:
Optimize flow

PATH 2:
Include feedback

PATH 3:
Learn from system

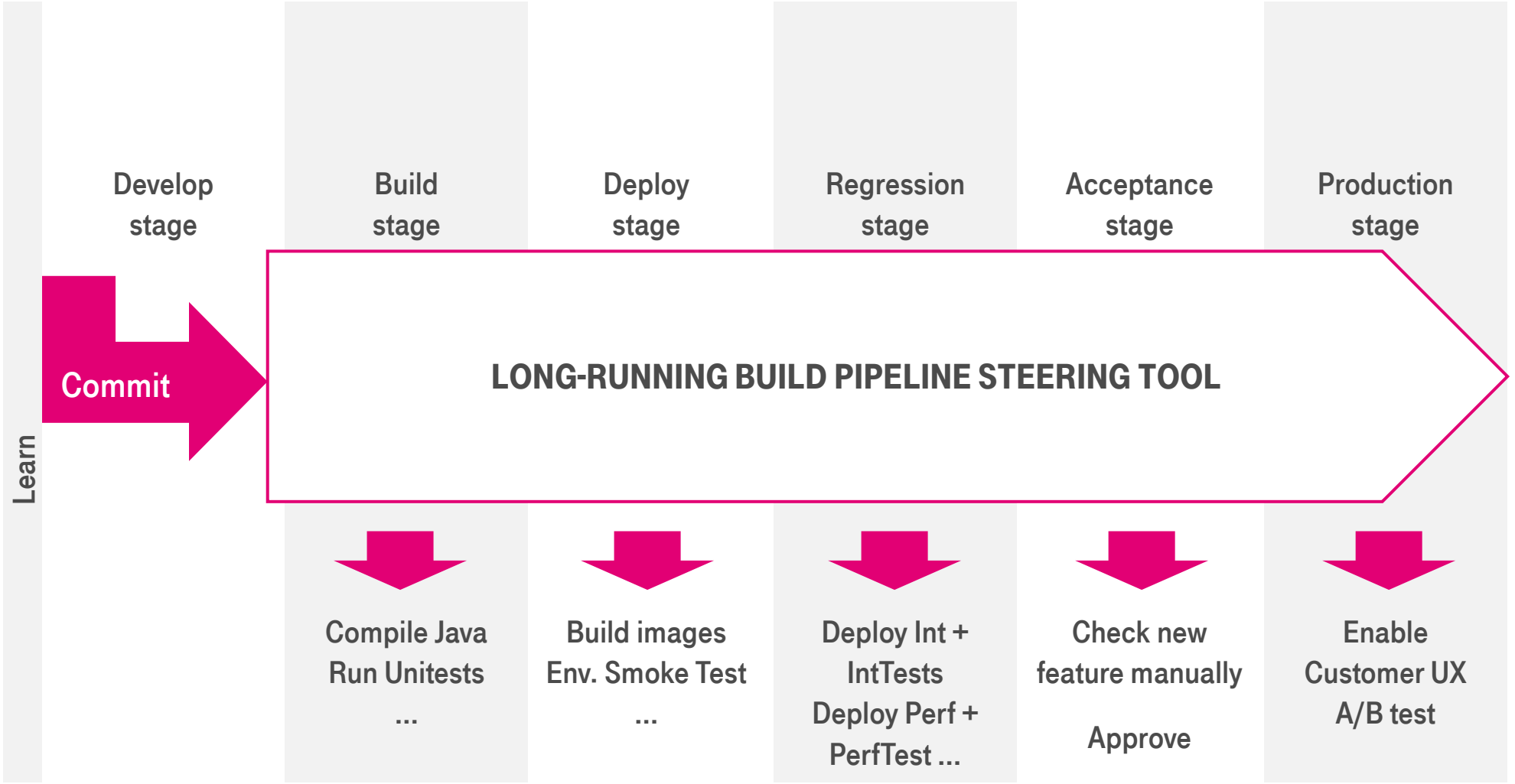
T · · Systems ·

DEVOPS PIPELINE TRIGGER: FEATURE CHECK IN

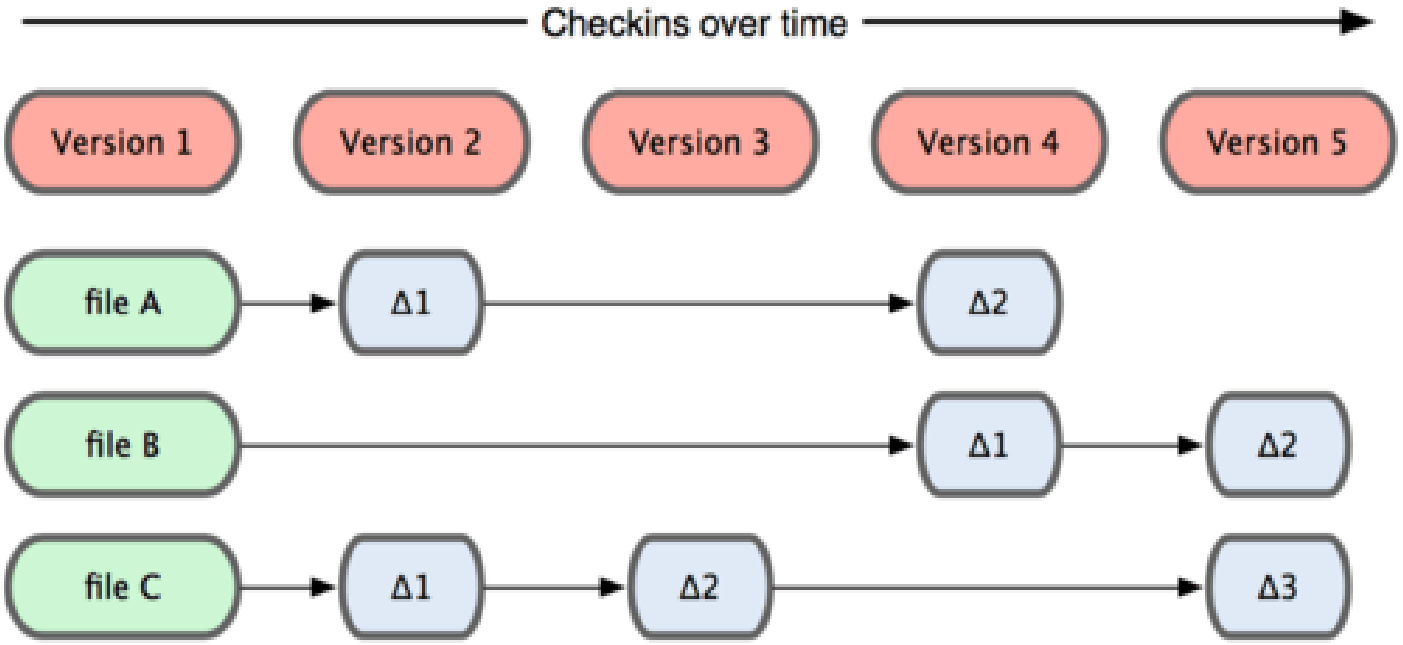


PIPELINE CONTROL

JENKINS, CLOUDBEES, CONCOURSE, OPENSHIFT, ...



KONZEPT VON GIT
DATENMODELL BEI CVS UND SVN



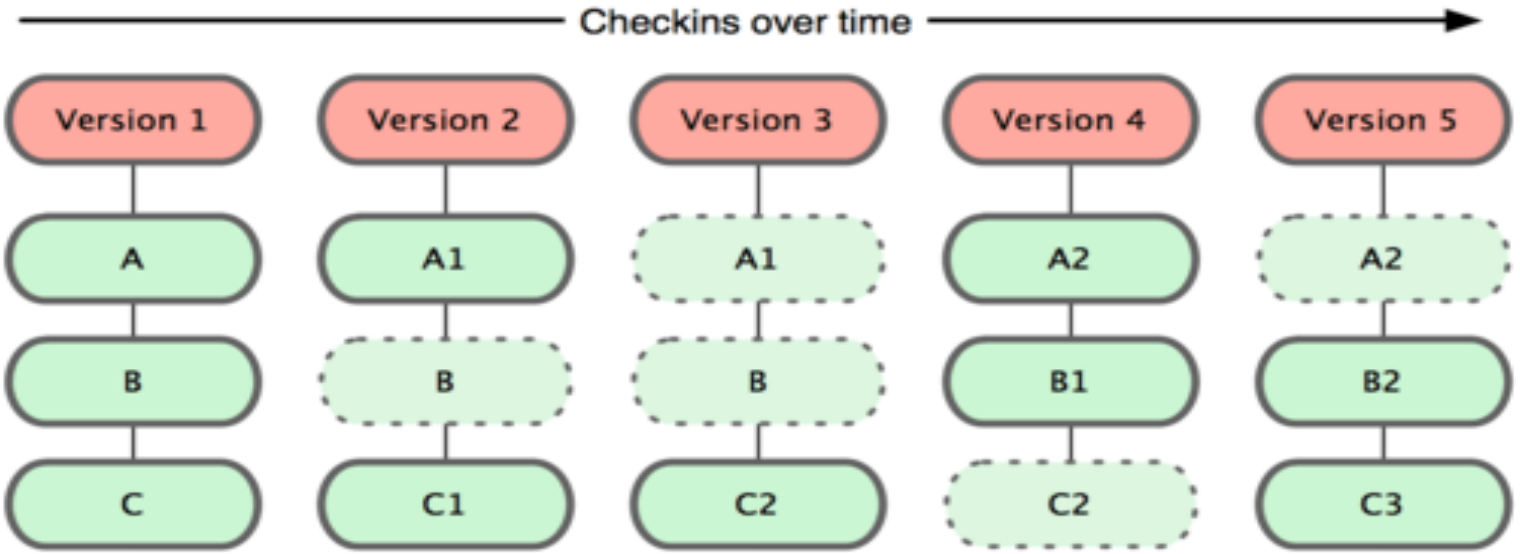
CVS und SVN speichern Informationen als eine fortlaufende von Änderungen an **Dateien.**

T · · Systems ·

KONZEPT VON GIT
DATENMODELL BEI GIT

Speicherung von Snapshots, nicht Diffs

- Git speichert Informationen als eine Reihe von Snapshots eines Mini-Dateisystems
- Bei jedem Commit wird der Zustand sämtlicher Dateien in diesem Moment gesichert und als Referenz auf diesen Snapshot gespeichert
- Git speichert Daten als eine Historie von Snapshots des Projektes



T · · Systems ·

BEREICHE EINES GIT PROJEKTES

ARBEITSVERZEICHNIS

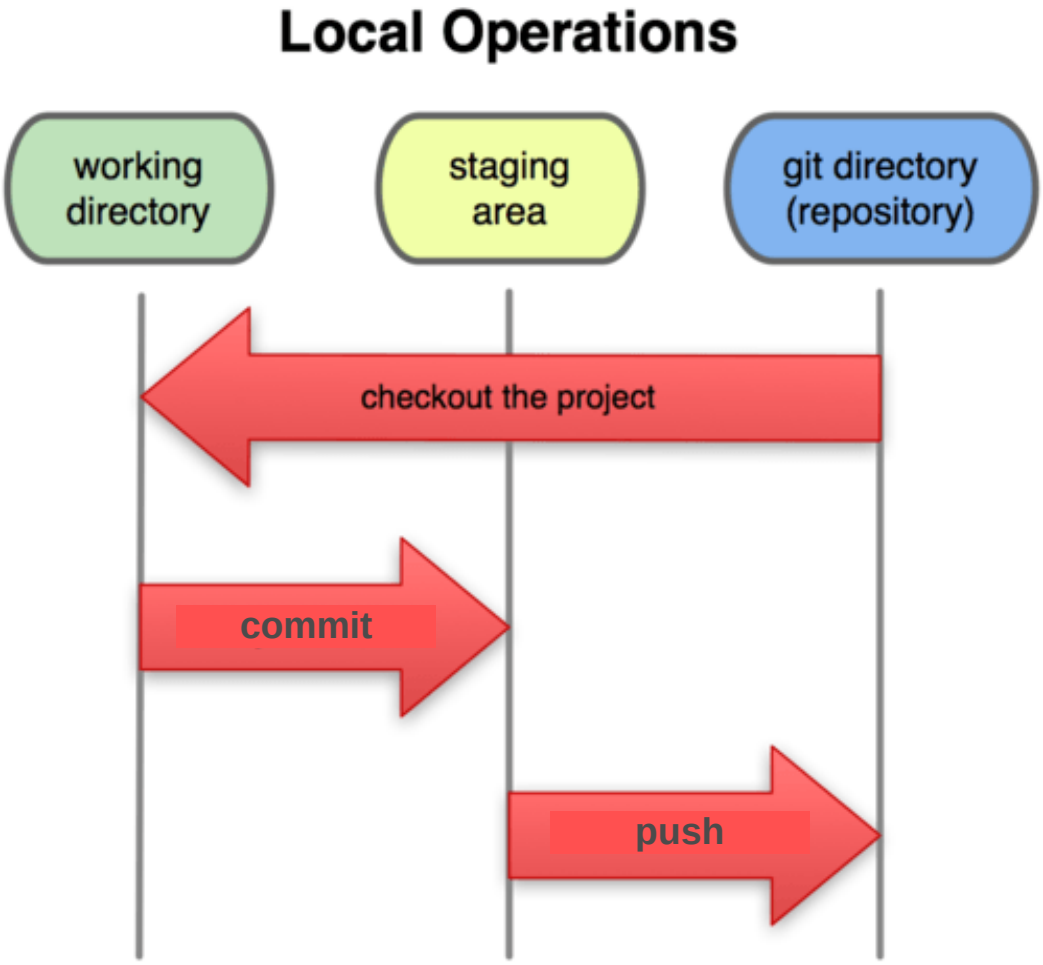
- Verzeichnis auf Festplatte zum Bearbeiten von Dateien

STAGING AREA (INDEX)

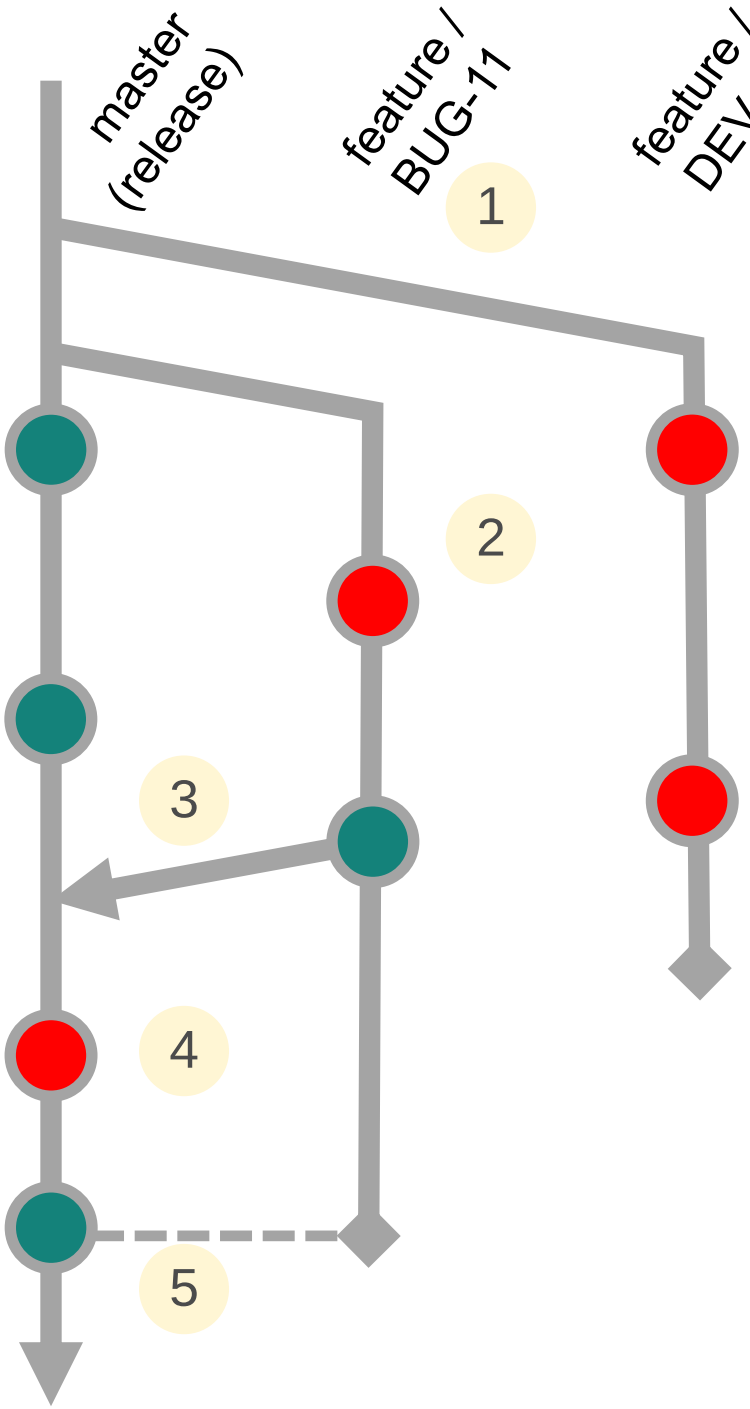
- Datei hält Informationen zu vorgemerkten Dateien

GIT VERZEICHNIS (REPOSITORY)

- Git Metadaten, lokale DB



T · · Systems ·



CONTINUOUS DELIVERY BRANCHING STRATEGY

1 **Every change is a feature branch.**
Branch name contains reference to backlog or issue.

2 **Break stuff on branches – check with Continuous integration**
Every checkin in branch triggers CI pipeline.

3 **Ready for release**

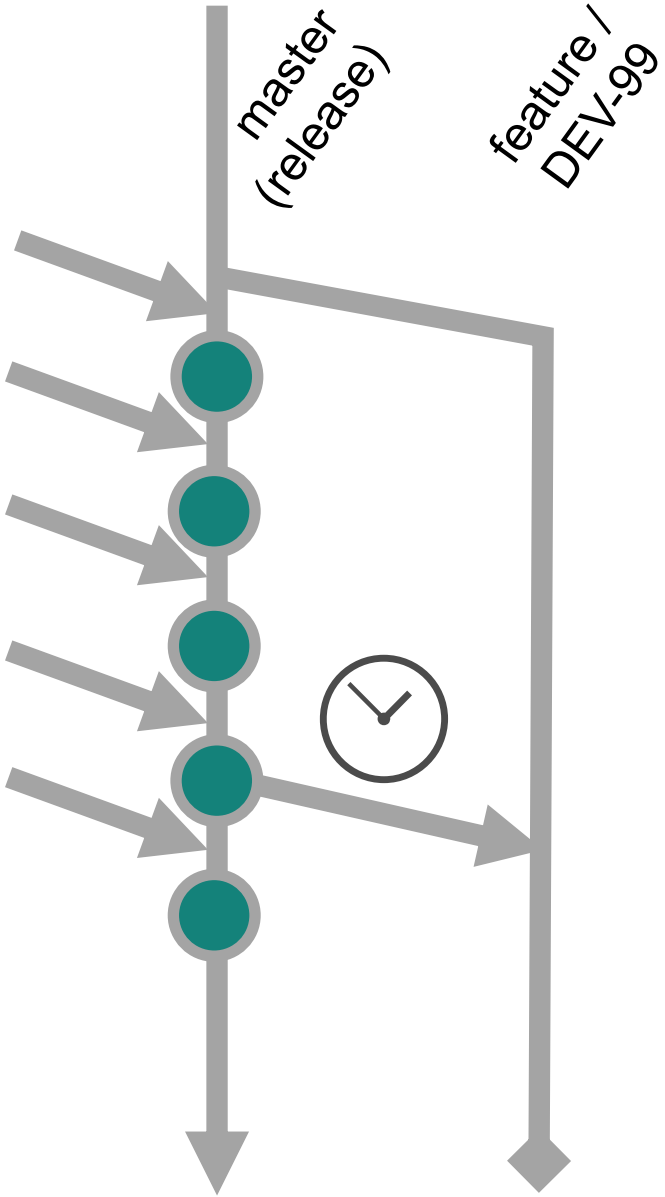
Variant a: Pull request
Reviewer pulls requests –
Continuous Delivery pipeline starts on approval

Variant b: Merge up release
Continuous delivery starts on merge to main branch

4 **Jidoka – „stop the line production“ on broken pipeline**
All people ready for rescue!

5 **Remove/cleanup branch after release**

T · · Systems ·



ANTI-PATTERN: LONG-RUNNING BRANCHES

Plan/split work
in short-running branches.

Avoid rebase!
Frequently merge master → feature

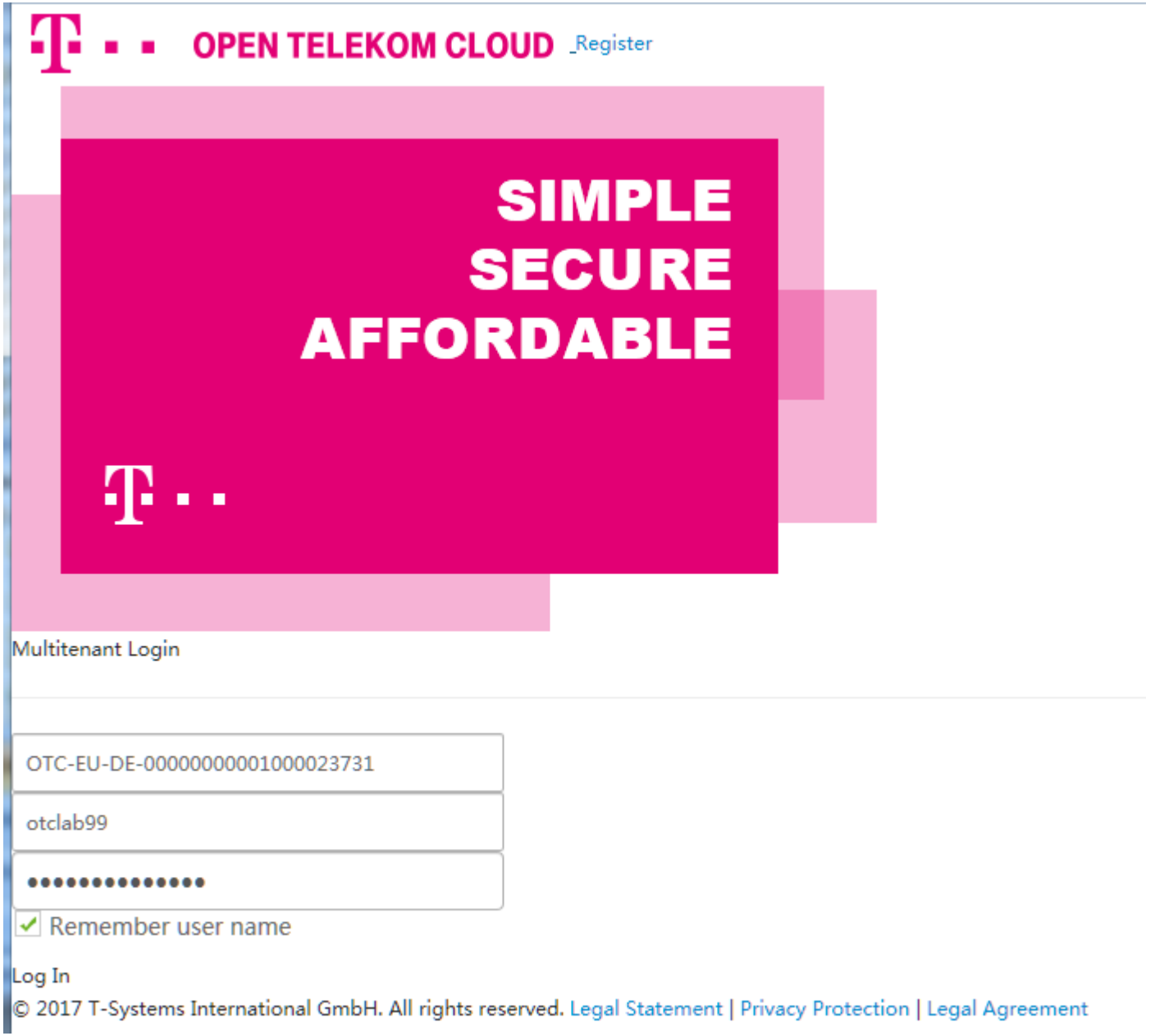
Useful reporting:

- all non-released branches with age
  → force merges in dailies
- Top-ten of long-running features/issues
  → input for backlog grooming &
  work planning (WIP)

https://www.atlassian.com/continuous-delivery/why-git-and-continuous-delivery-are-super-powered

EXERCISE 1:
PRODUCTION PIPELINE

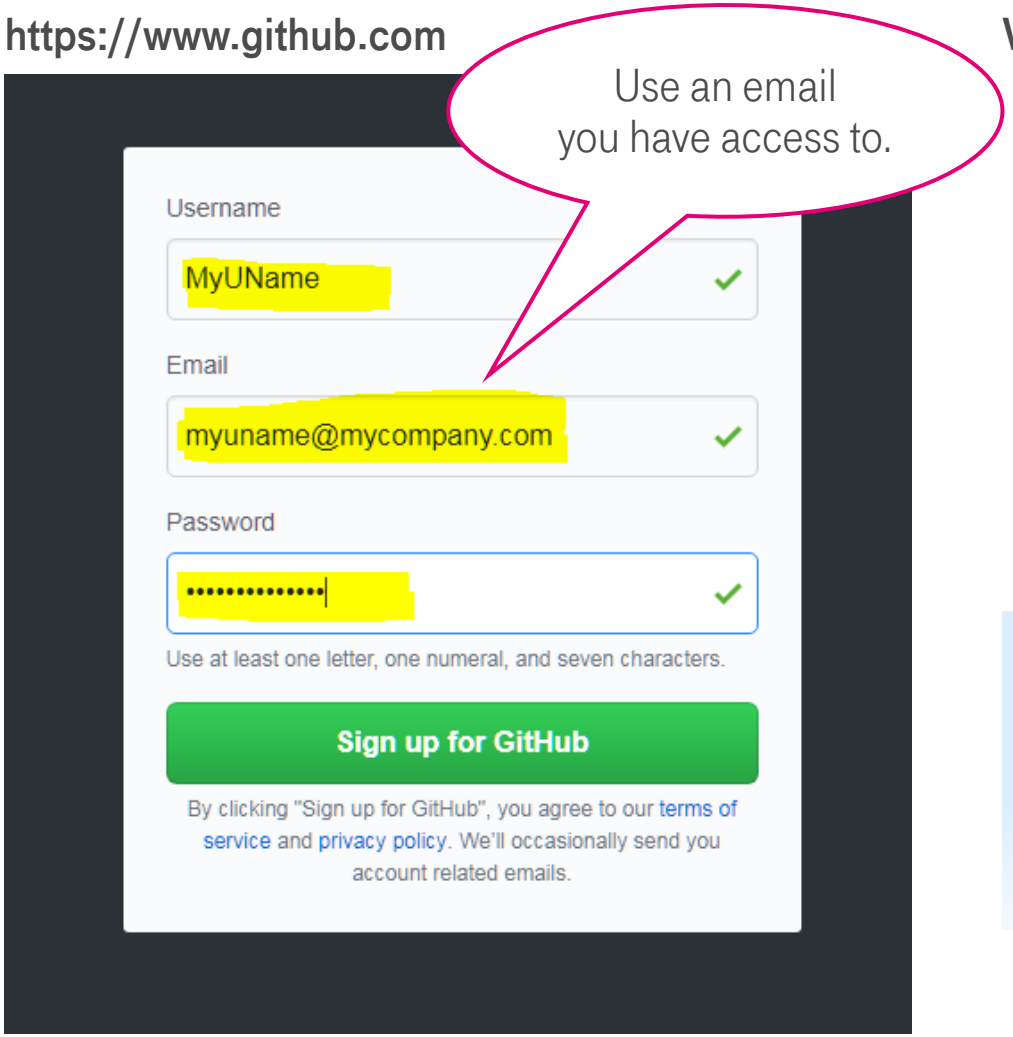
CONSOLE.OTC.T-SYSTEMS.COM



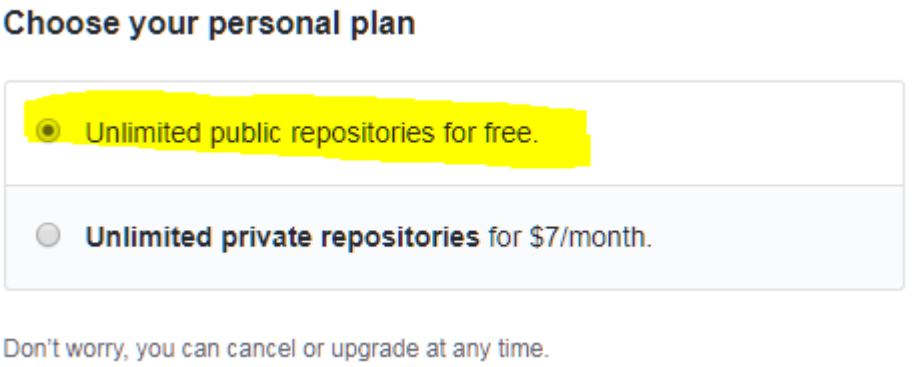
T · · Systems ·

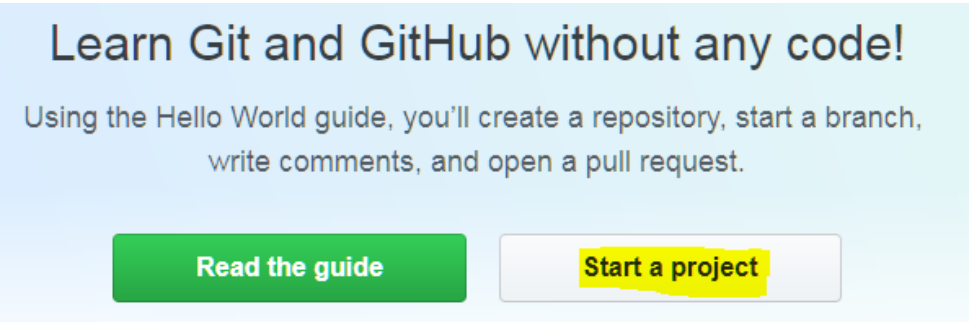
REGISTER A PUBLIC GITHUB ACCOUNT

https://www.github.com



Verify your email address: Klick verification link

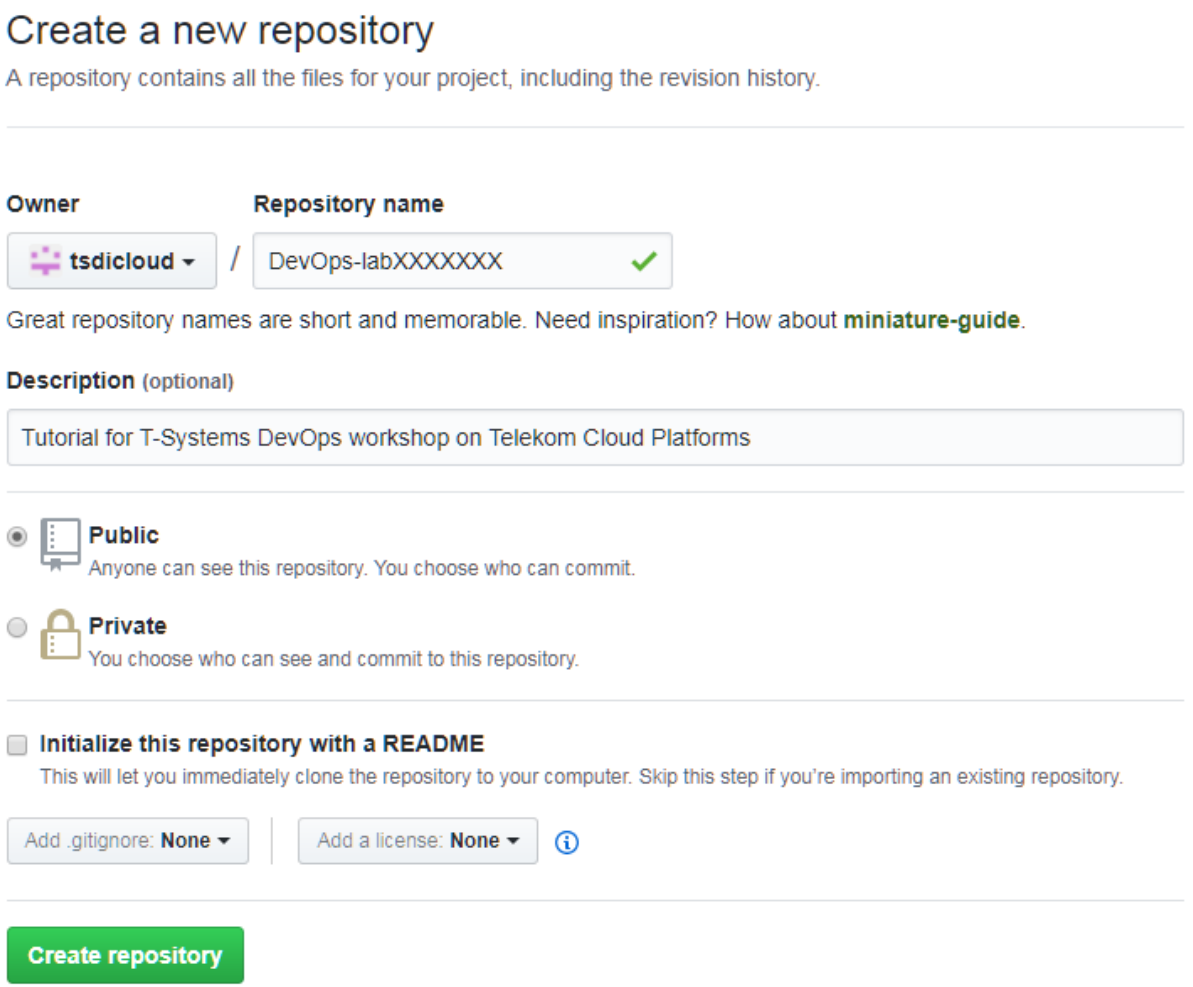




T · · Systems ·

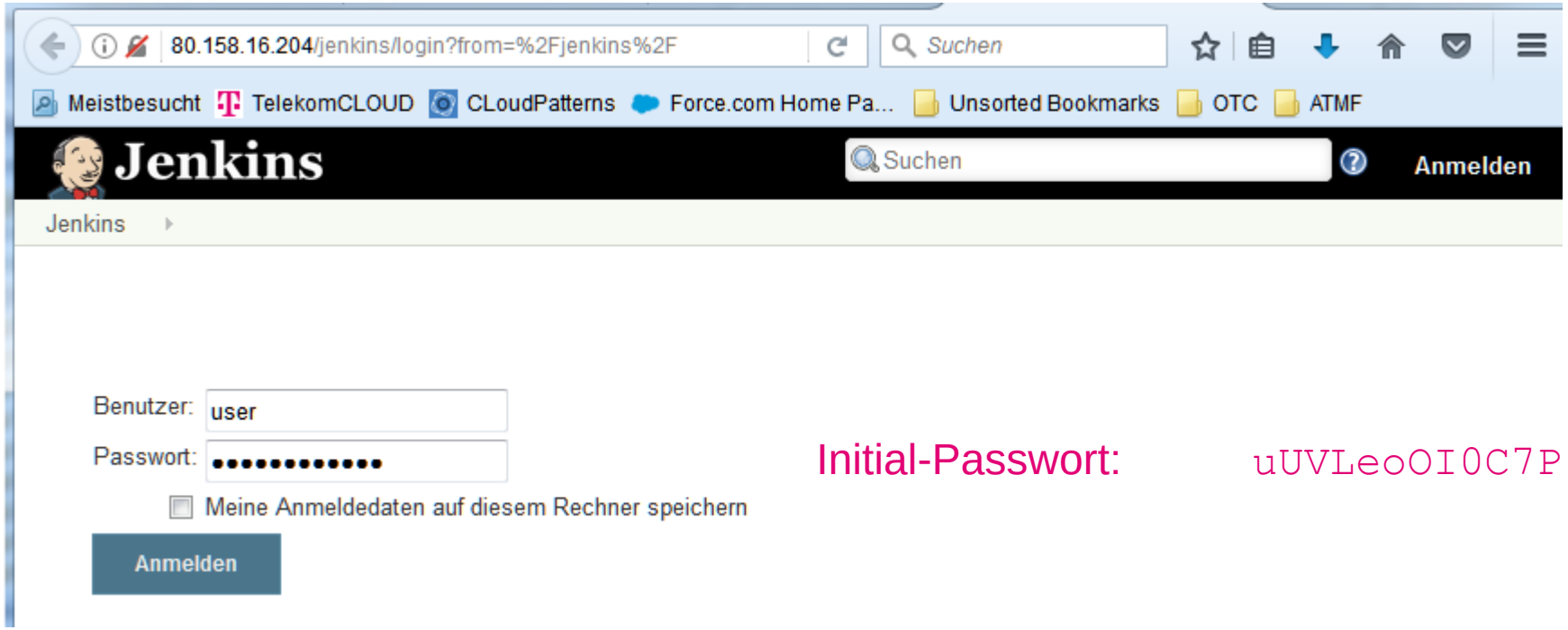
CREATE A (TEMPORARY) GITHUB PROJECT

CHOOSE YOUR OWN ORG AND REPO NAME



T · · Systems ·

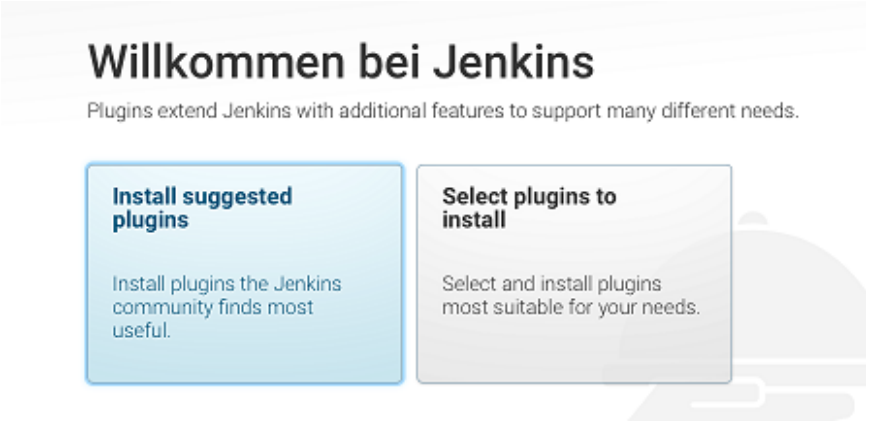
JENKINS(1): LOGIN



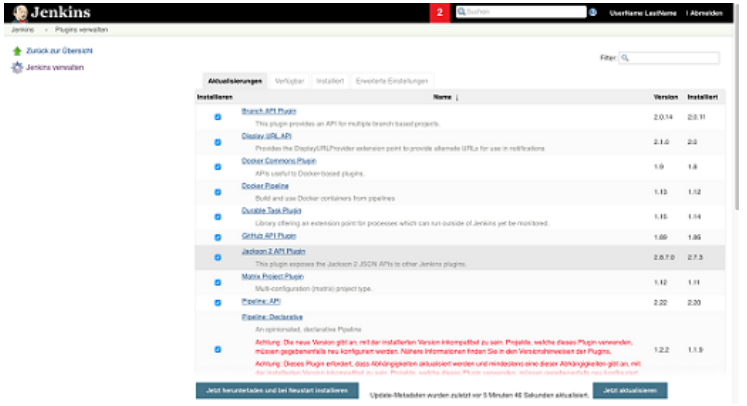
Initial-Passwort: `uUVLeoOI0C7P`

SETUP JENKINS

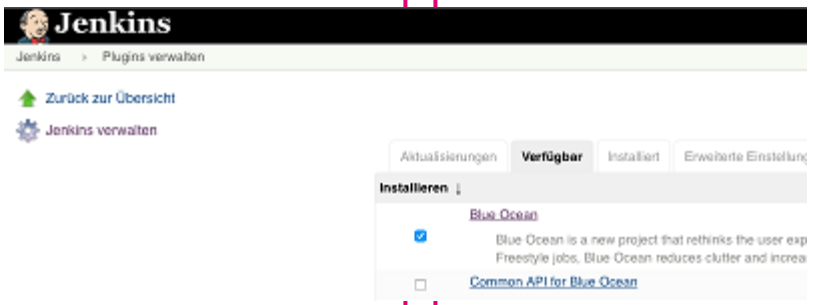
(1) Install default plugins + restart



(2) Update default plugins to current state + restart

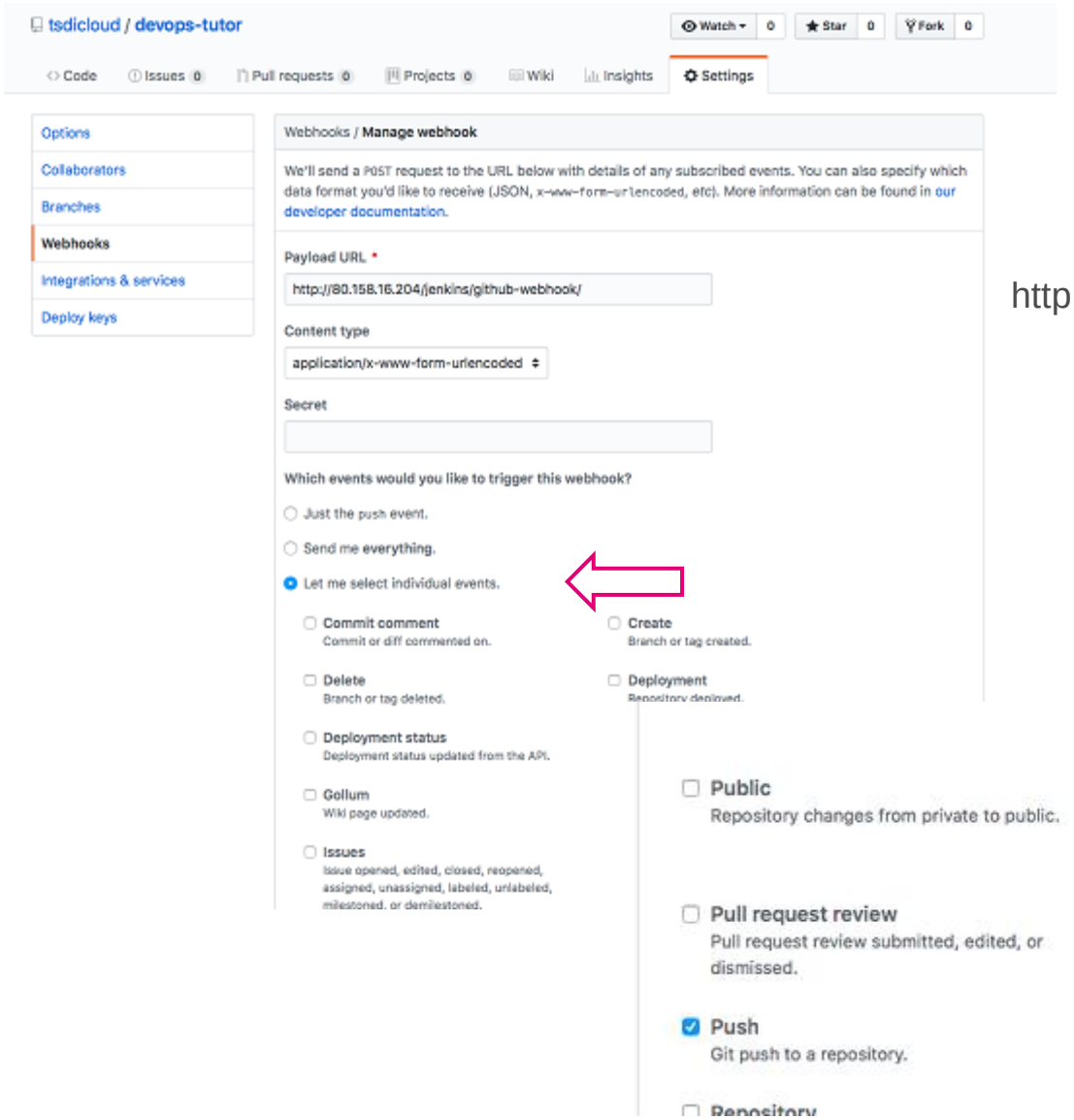


(3) Install GitHub integration plugin



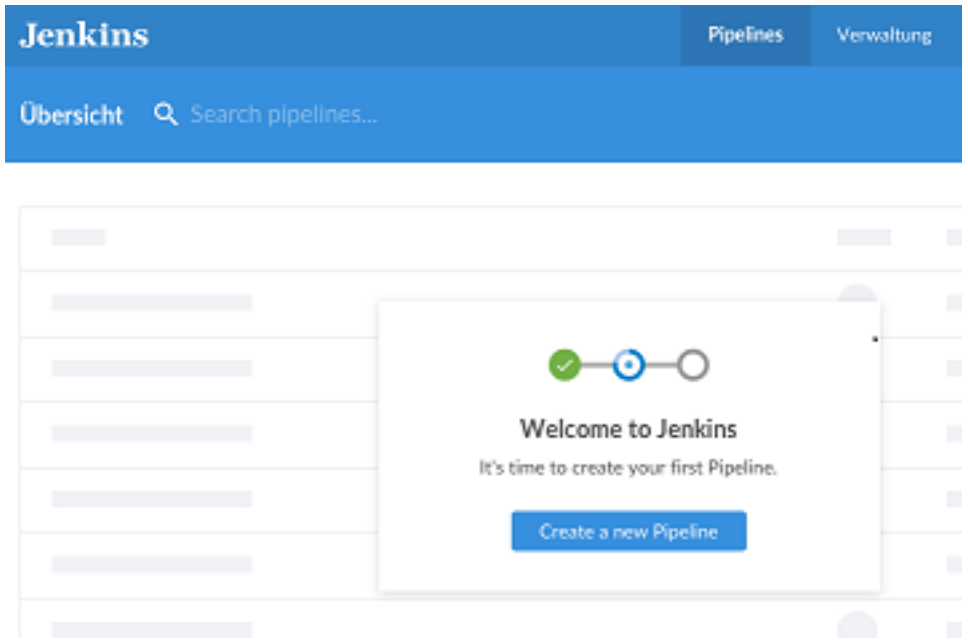
(4) Install Blue Ocean plugins + restart

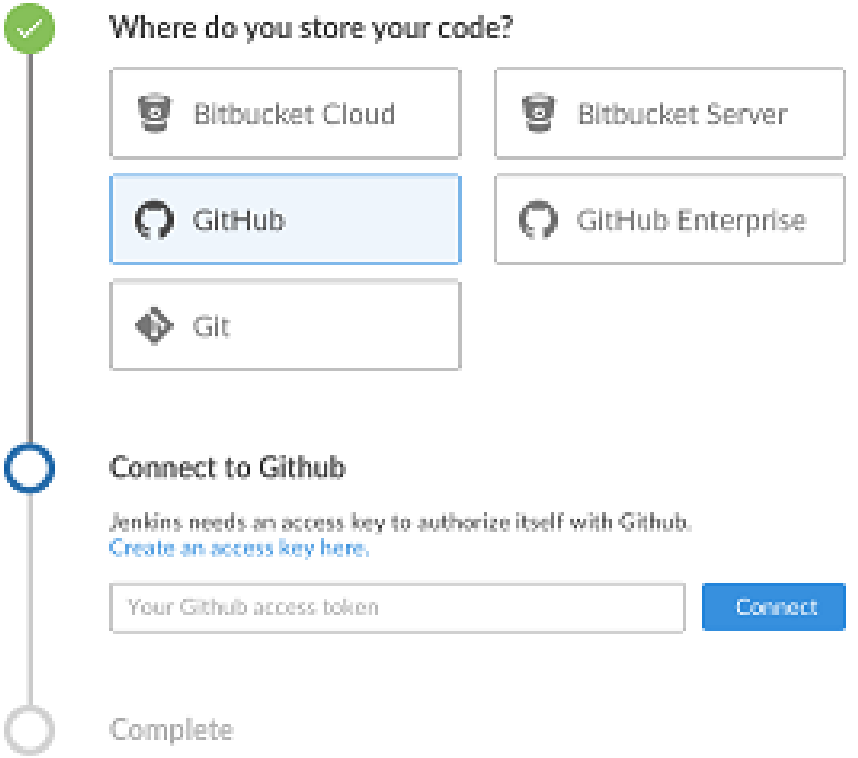
SETUP WEBHOOK IN GITHUB



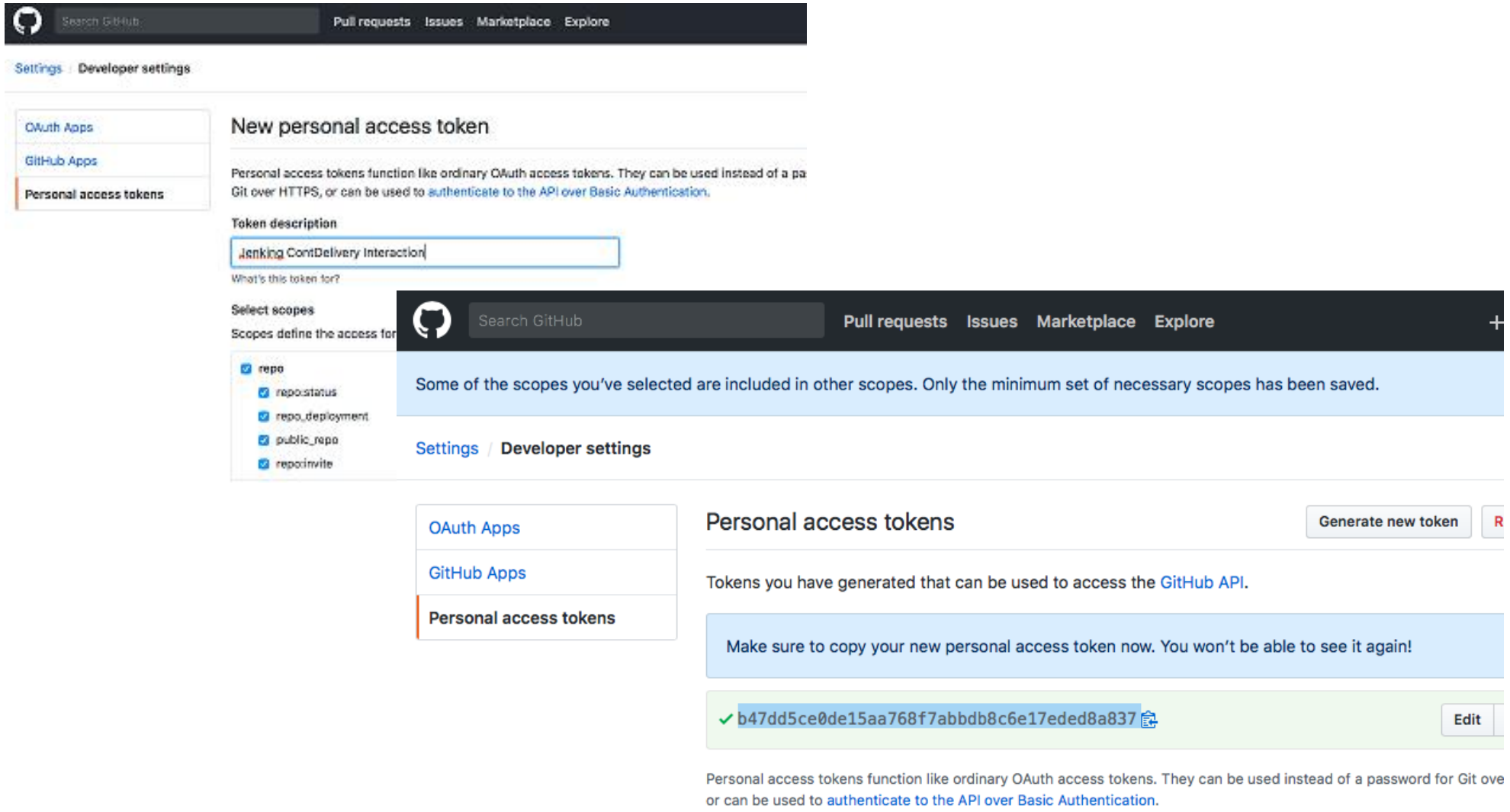
http://<jenkins-eip>/jenkins/github-webhook/

CREATE FIRST PIPELINE(1)

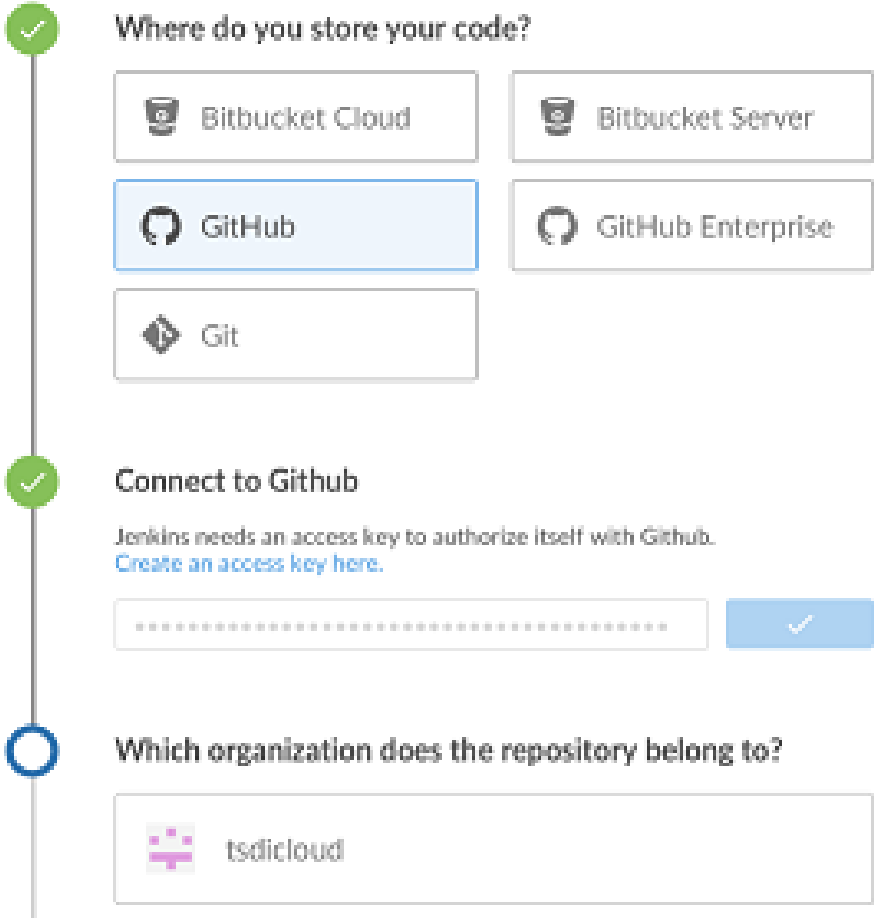


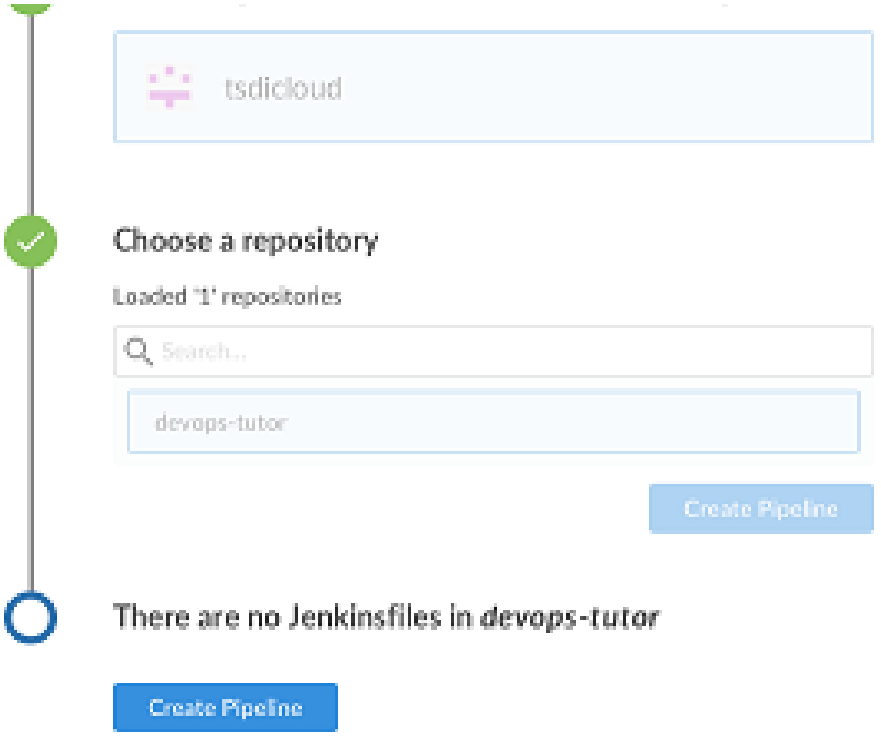


CREATE PIPELINE(2)
COPY TOKEN FROM GITHUB



CREATE PIPELINE(3) – USE ORG AND REPO FROM BEFORE





DEMO PIPELINE INSTALLATION

Laptop local: Get demo projects

```
$ git clone https://bitbucket.org/gdccloudconsulting/devops_lab
```

Laptop local: Get GitHub project

```
$ git clone https://github.com/<owner>/<repository_name>
```

Laptop local: Copy Jenkinsfile

```
$ cp devops_lab/exercises/jenkins/Jenkinsfile <repository_name>
$ cd <repository_name>
$ git commit Jenkinsfile -m „Example pipeling with CI and CD support“
$ git config user.name <github username>
$ git config user.email <email for github notifications>
$ git push origin master
```

GIT BASICS – CONSISTENT CHECKIN COMMENTS

GIT cheat sheet:
https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf

How to Write a Git Commit Message
https://chris.beams.io/posts/git-commit/

TIP: Complete the sentence

"If applying this patch,
the change will...
<verb> <object of change>"

```
$ git log --oneline -5 --author pwebb --before "Sat Aug 30 2014„
5ba3db6 Fix failing CompositePropertySourceTests
84564a0 Rework @PropertySource early parsing logic
e142fd1 Add tests for ImportSelector meta-data
887815f Update docbook dependency and generate epub
ac8326d Polish mockito usage
```

T · · Systems ·

START DEVELOPMENT AND CONTINUOUS INTEGRATION

Laptop local: Create feature branch

```
$ cd <repository_name>
$ git branch feature/DEV-4711
$ git checkout feature/DEV-4711
```

Add some file „componentA.txt“ to the branch

```
$ git commit componentA.txt –m „Add component A initial implementation“
$ git push origin feature/DEV-4711
```

CI pipeline in Jenkins should start.

START CONTINUOUS DELIVERY – DIRECT MERGE

Laptop local: Switch to master branch, merge feature branch to it

```
$ cd <repository_name>
$ git checkout master
$ git merge feature/DEV-4711
```

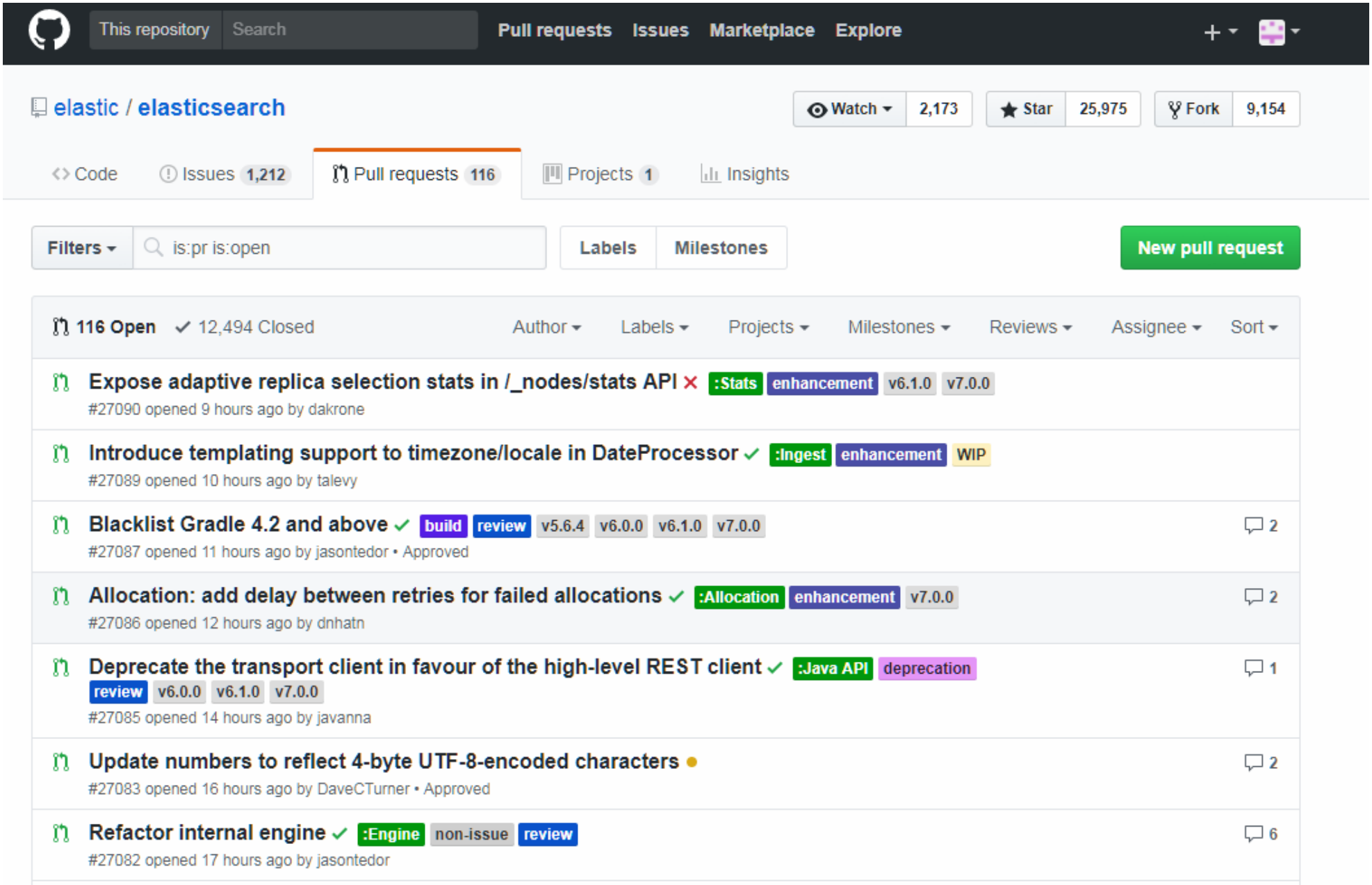
Push new master version

```
$ git commit componentA.txt –m „Add component A initial implementation“
$ git push origin master
```

CD pipeline in Jenkins should start.

Accept or cancel pipeline to finish CD job.

PULL REQUEST (REAL WORLD PROJECT)




KAFFEE-
PAUSE

AGENDA

01	Intro: Continuous Delivery Pipeline revisited
02	Reproducibility: Everything as Code
	02.1 Exercise: Trigger pipeline (Jenkins from GIT)
03	**Provisioning time: Infrastructure as Code**
	03.1 Exercise: Automated install and configuration
04	Dev-Prod-parity: Immutable Infrastructures
	04.1 Exercise/Video: Image bakery
05	Conways law and continuous feedback
	05.1 Exercise: ELK logging as microservices
06	Learn: Red/black, A/B and canaries
	06.1 Exercise: My little A/B test

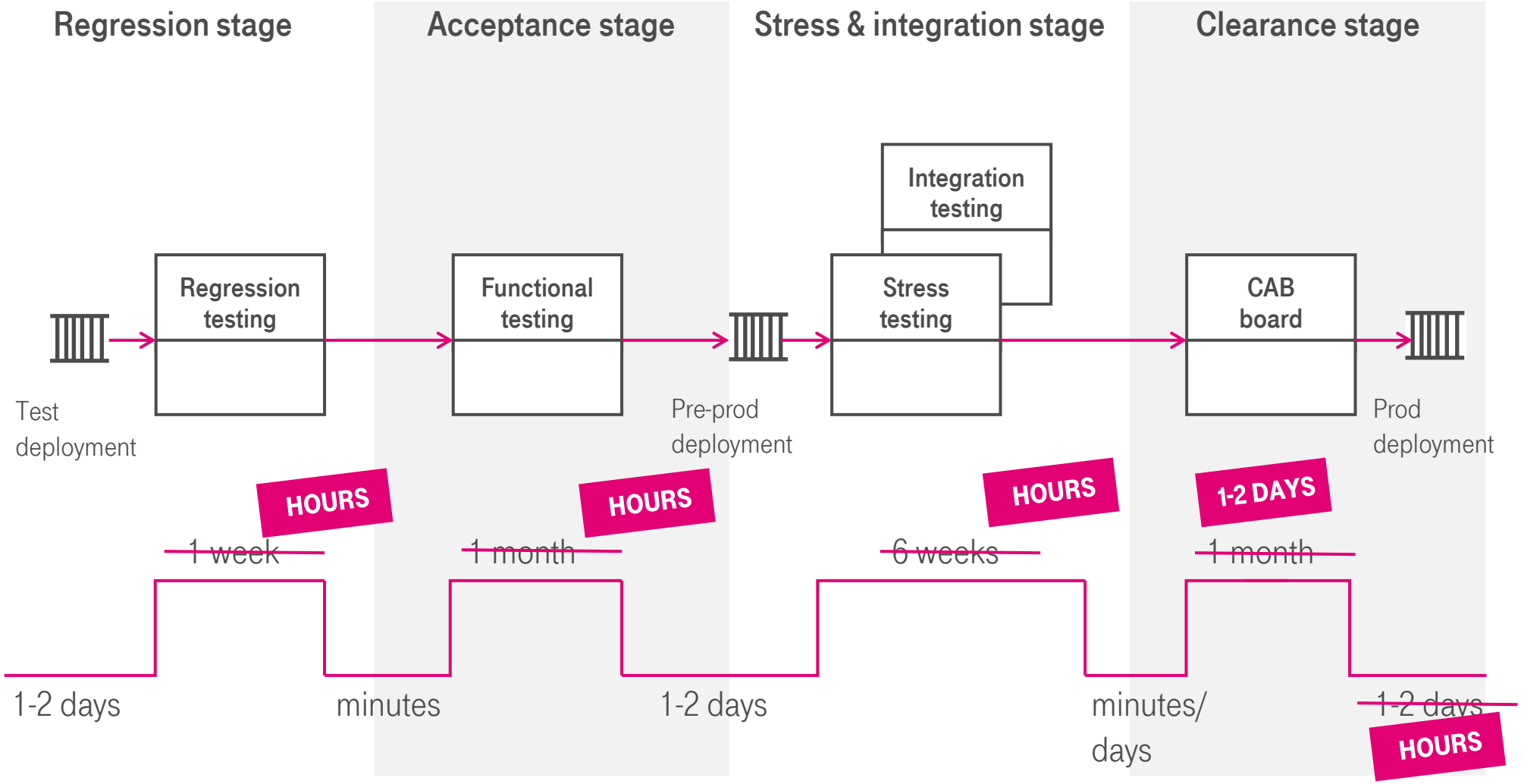
PATH 1:
Optimize flow

PATH 2:
Include feedback

PATH 3:
Learn from system

T · · Systems ·

VALUE STREAM ANALYSIS (TEST ONLY, SIMPLIFIED)



T · · Systems ·

SMALL BATCH PRODUCTION
THE ENVELOPE EXPERIMENT

JAMES WOMACK AND DANIEL JONES, LEAN THINKING

FATHER

- put address on envelope
- put stamp on envelope
- put letter into envelope
- seal envelope

Next

SON

- all addresses on all envelope
- all stamps on all envelope
- all letter into envelopes
- seal envelopes

→ **Forgotten steps: sorting, stacking, handling**

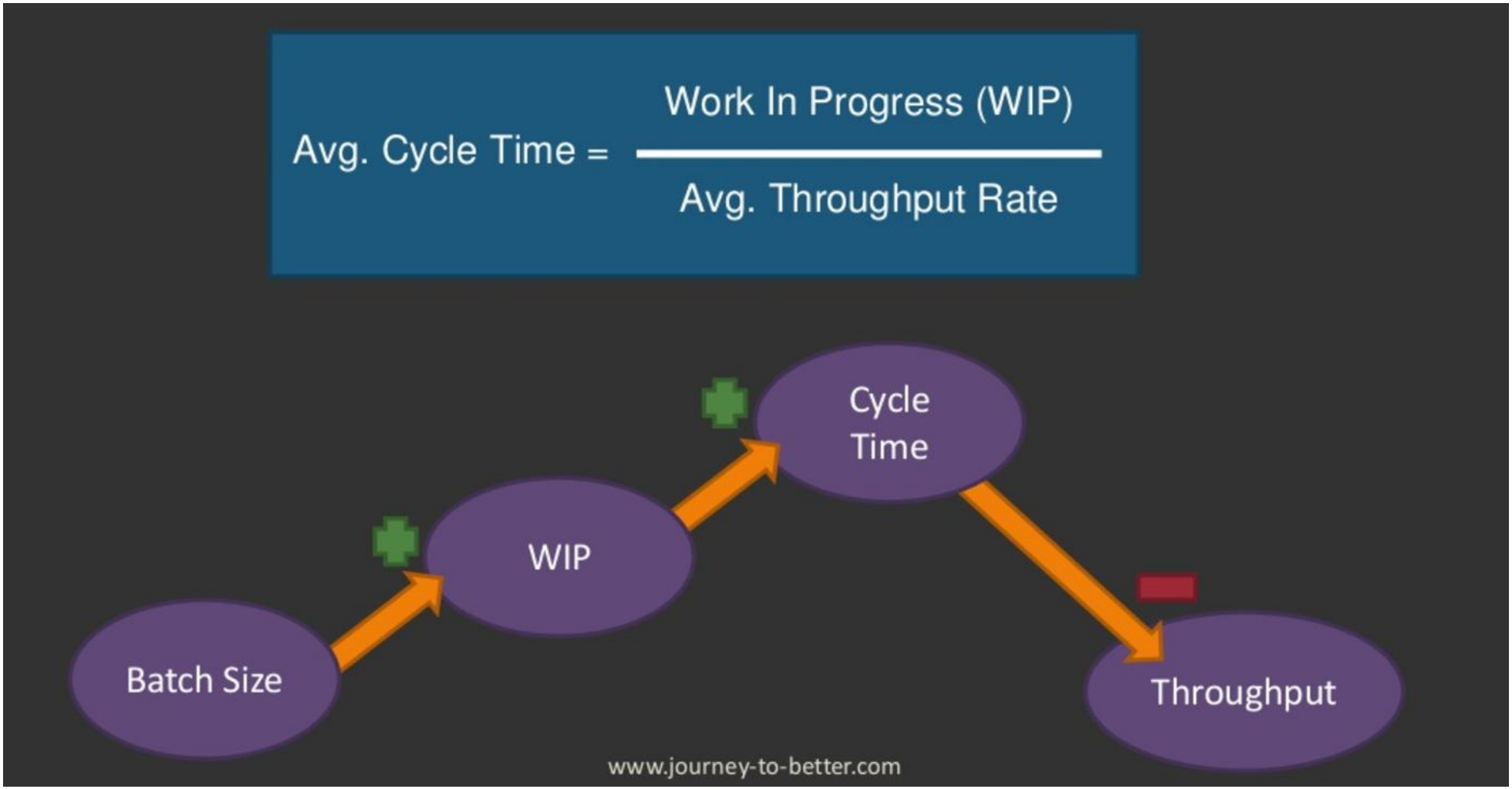
→ **Learning effect is overestimated**

Father´s strategy wins (empirically proven)

AND is interruptable

AND has fewer risk for throughput drops.

QUEUING THEORY: LITTLES LAW



https://de.slideshare.net/andrewrusling/improving-throughput-with-the-theory-of-constraints-and-queuing-theory

FINDINGS

Work storage = high WIP → high cycle time → low throughput

→ Smaller batch reduces service time.

Utilisation per service station is maximal at the bottleneck.

→ Keep bottleneck at maximal speed.

More ressouces increase throughput at a work station.

→ More ressources only help if they scale the bottleneck.

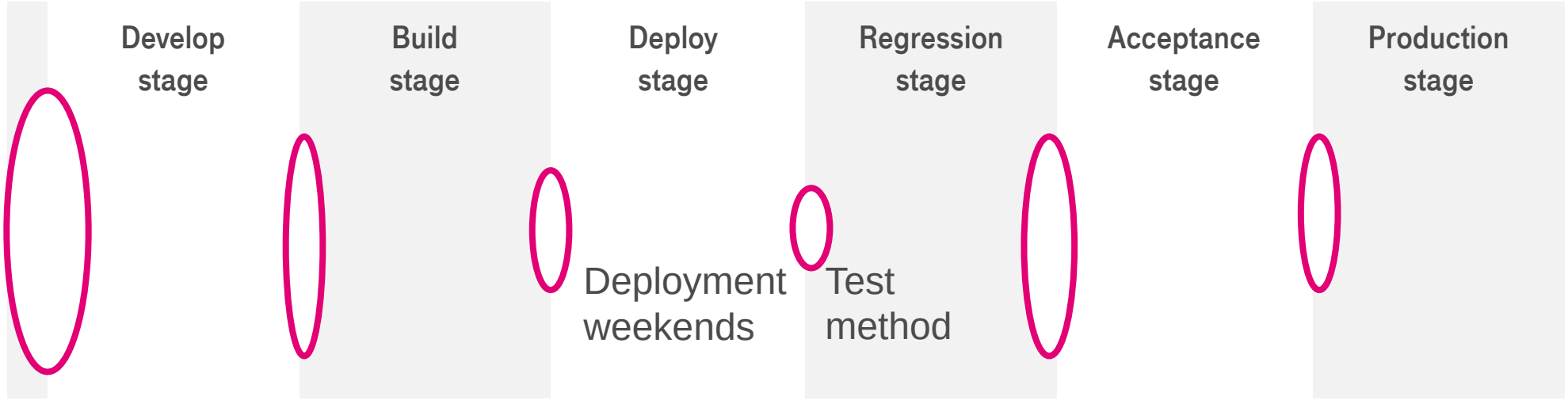
“

A chain is not stronger than its weakest link.

”

T · · Systems ·

CONSTRAINT OPTIMISATION
HUNT FOR BOTTLENECKS



- Finding: Throughput limitation by MANUAL regression testing
  Measure: 90% automation of regression tests
- Finding: Throughput limitation by MANUAL environment deployments
  Measure: Automated deployment
- Finding: Troughput limitation by AVAILABILITY of required environments
  Measure: ALL ENVIRONMENTS IN CLOUD; add Infrastructure as code to automation

“

If its hard:

leave it or make it routine.

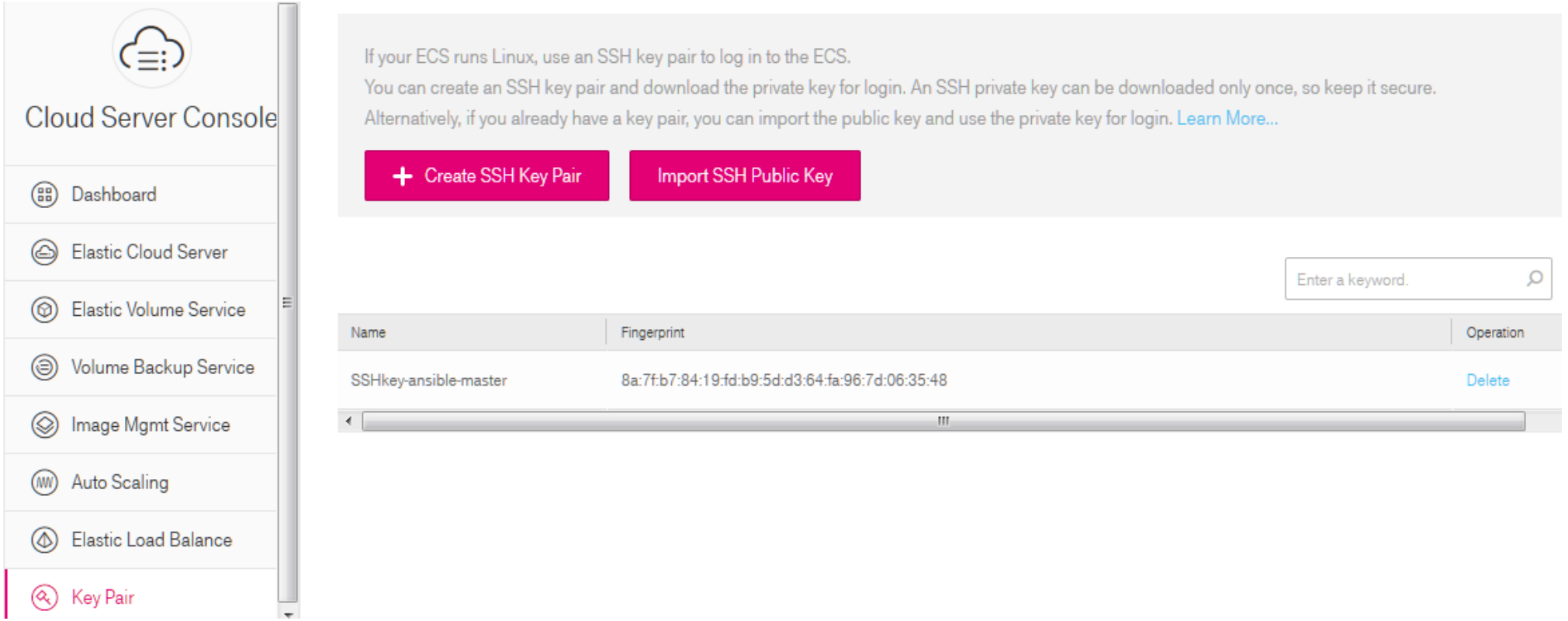
Kurt Garloff

”

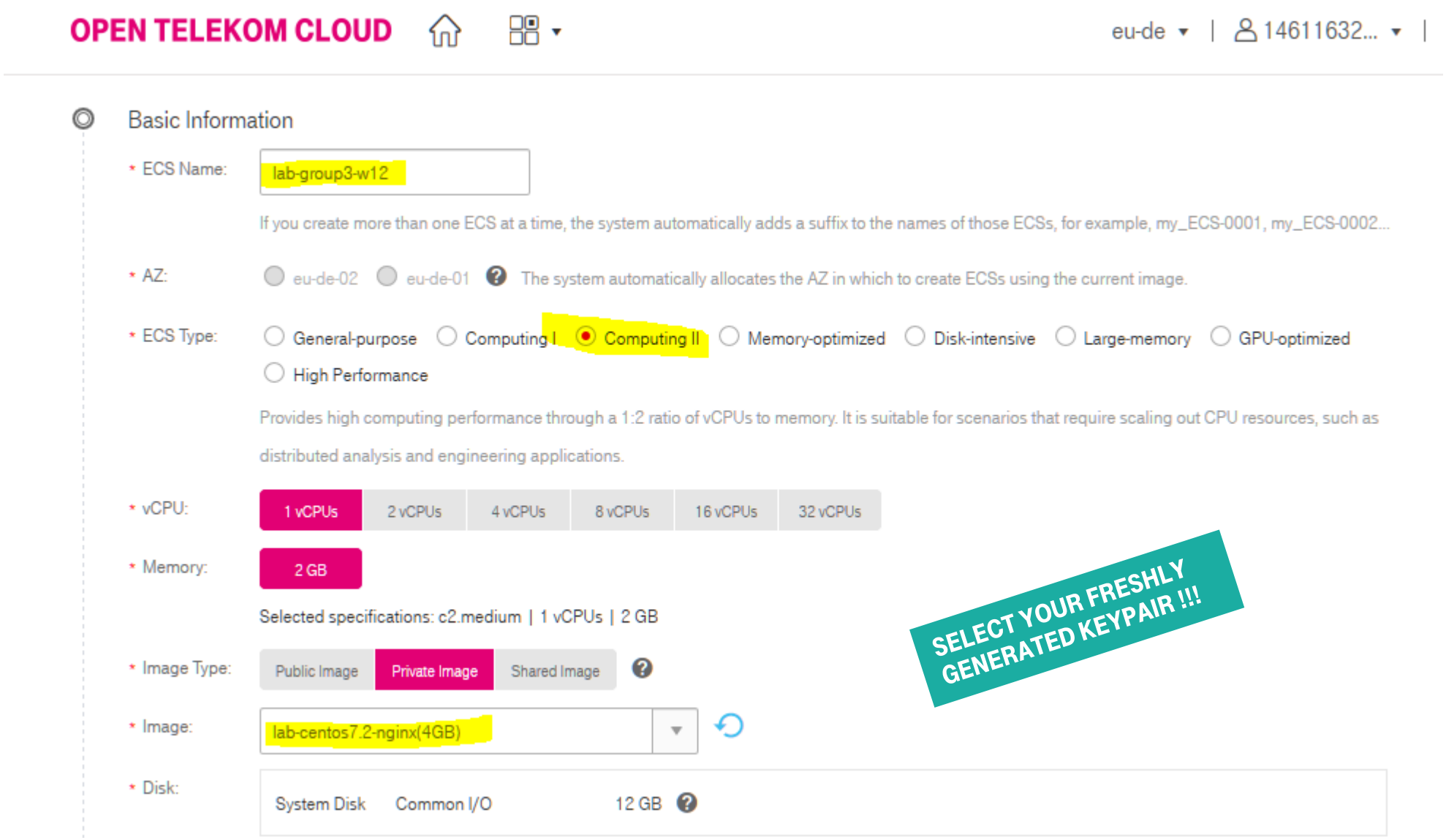
T · · Systems ·

EXERCISE 2: AUTOMATED INSTALL AND CONFIG

02, STEP 1: PREPARE KEY PAIR



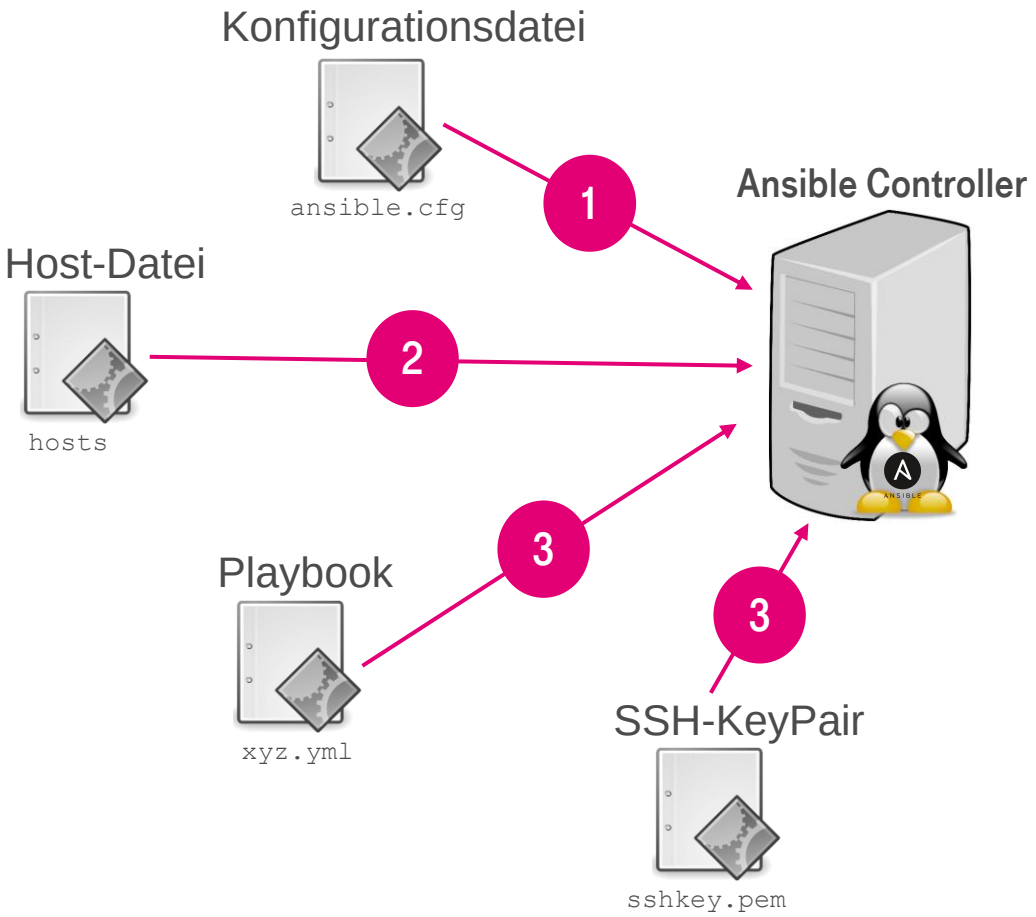
02, STEP 2: CREATE A PRIVATE SERVER



T · · Systems ·

ANSIBLE – AGENTLESS CONFIGURATION MANAGEMENT

THE „SWISS ARMY KNIFE“ FOR OPS AUTOMATION



T · · Systems ·

03, STEP 1: INTIAL LOGIN TO MASTER SERVER
(ONLY FOR LAB, USUALLY NOT REQUIRED)

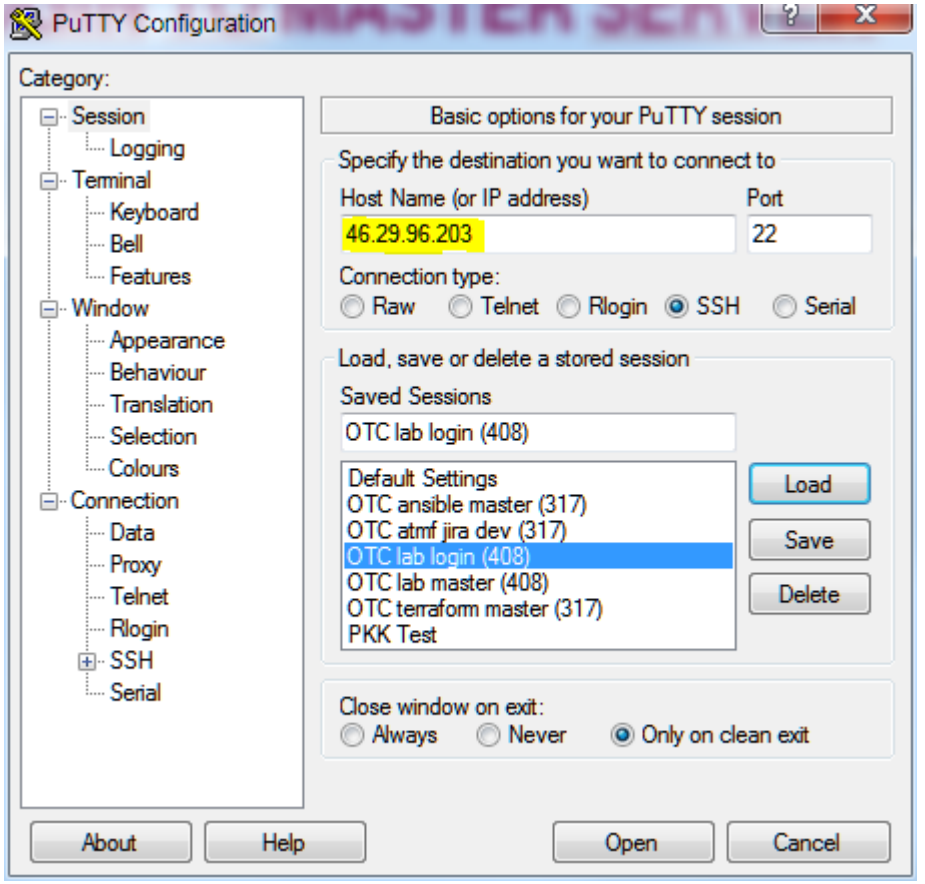
Server IP: `46.29.96.203`

User: `otclabX (see card)`

Pass: `<published in classroom>`

Note: If using VPN, you may have to use a proxy (http)
Win/putty: under `Connection>Proxy`
Mac/Lin: `ssh -l otclabX 46.29.96.203`

Note2: if connection breaks occur :
Win/putty: under `Connection`
seconds between keepalives
Mac/Lin: `-o ServerAliveInterval=10`




```
.ssh/config:
ProxyCommand nc -X connect -x proxyhost:proxyport %h %p
ServerAliveInterval 10
```

T · · Systems ·

03, STEP 2: EXTRACT PUBLIC/PRIVATE KEYS FOR SSH

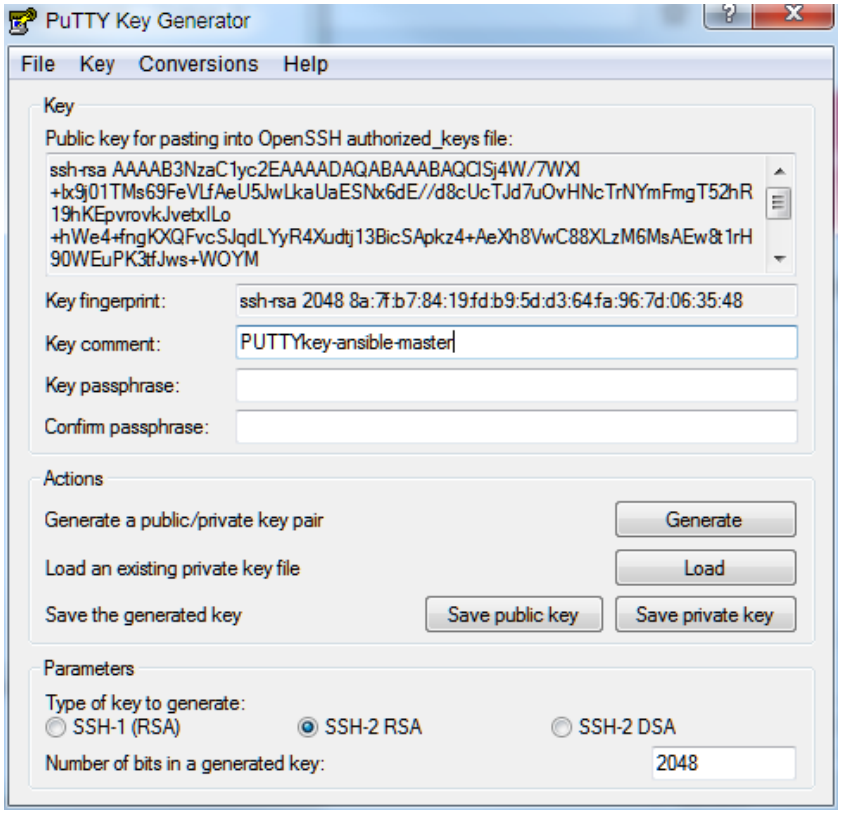
MAC OS (login user is „otclabX“):

```
ssh-keygen -y -f SSHkey-otclabX.pem > SSHkey-otclabX.pub
```

(use output directly, add comment)

Windows / PUTTYgen / Putty: Edit public key

- remove heading/trailing line with `---`
- add ssh-rsa at the beginnning
- remove linebreaks
- add a comment

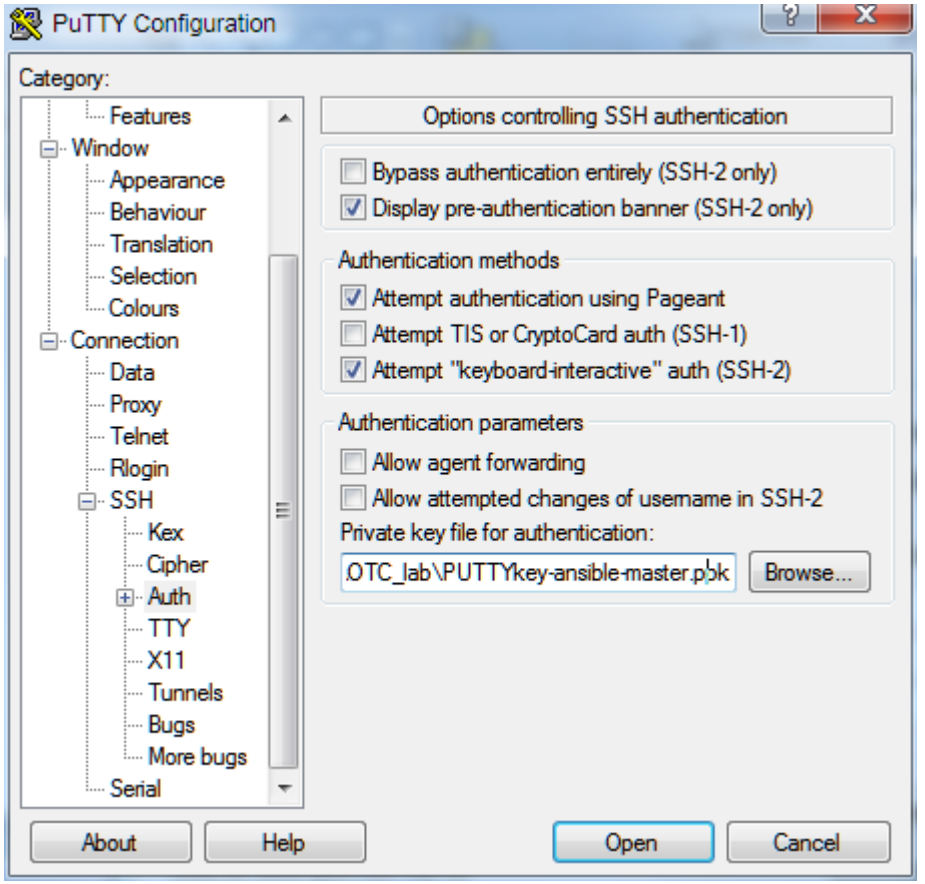


T · · Systems ·

01, STEP 3: PREPARE ACCOUNT ON MASTER
(ONLY FOR LAB, USUALLY NOT REQUIRED)

- **Copy** `SSHkey-otclabX.pub` **content to** `.ssh/authorized_keys`
- **Login via putty / SSH with given key – should work**
  Do not forget to set login user in Putty to otclabX
- **Copy private key** `SSHkey-otclabX.pem` **to** `.ssh/id_rsa` **with sftp for private server access**

This is an easy preparation to connect private servers from here



Tests:

1. `ssh -l linux 10.128.0.x`
2. `curl http://10.128.0.x`

T · · Systems ·

01, STEP 4: CLONE SOME EXAMPLE FILES, EDIT BASHRC

```
git clone https://bitbucket.org/gdccloudconsulting/devops_lab.git
```

TASKS

- **Copy** `openstack/otc_certs.pem` **to home top directory.**
- **Copy environment variables from** `bashrc-example.sh` **to** `$HOME/.bashrc`
- **Adapt variables, use the API key from your card**

T · · Systems ·

PLAYBOOKS, YAML, JINJA2

- **Format:**

```
---
key1: value1
key2: "{{ jinja2_var }}"
  subkey1: subvalue1
```

indent 2 spaces

```
-arrayvalue1
- arrayvalue2
```

- **Simple playbooks:**

```
---
- hosts: <inventory group>
  vars:
    variable1: value1
  tasks:
  - name:
```

T · · Systems ·

SIMPLE ACTION

EXERCISE: „SERVERNAME“

Add a server-identification string and an indivual color to the index.html file in your new server

INVENTORY + EXERCISE „SERVERNAME“

- Inventory file „hosts“:

```
[master]
localhost ansible_connection=local

[worker]
10.128.xx.xx
```

- Start playbook

```
> ansible-playbook -i hosts servername.yml
```

- Module documentation

http://docs.ansible.com/ansible/index.html

T · · Systems ·


MITTAG
ESSEN

AGENDA

01	Intro: Continuous Delivery Pipeline revisited	**PATH 1:** Optimize flow
02	Reproducibility: Everything as Code	
	02.1 Exercise: Trigger pipeline (Jenkins from GIT)	
03	Provisioning time: Infrastructure as Code	
	03.1 Exercise: Automated install and configuration	
04	**Dev-Prod-parity/Immutable Infrastructures**	
	04.1 Exercise: Image bakery	
05	Conways law and continuous feedback	**PATH 2:** Include feedback
	05.1 Exercise: ELK logging as microservices	
06	Learn: Red/black, A/B and canaries	**PATH 3:** Learn from system
	06.1 Exercise: My little A/B test	

T · · Systems ·

> *In the old way of doing things, we treat our servers like pets, for example Bob the mail server. If Bob goes down, it's all hands on deck. The CEO can't get his email and it's the end of the world. In the new way, servers are numbered, like cattle in a herd. For example, www001 to www100. When one server goes down, it's taken out back, shot, and replaced on the line.*
>
> ***Tim Bell, Cern***

T · · Systems ·

CLOUD CHARACTERISTICS REVISITED

SO MUCH THEORETICAL WORK IN THE PAST

National Institute of Standards and Technology
U.S. Department of Commerce

On-demand self-service	Broad network access	Ressource pooling	Rapid elasticity	Measured service

T · · Systems ·

“

Our “holy cow” pet is Dev→PreProd→Prod
expensive, static environments.

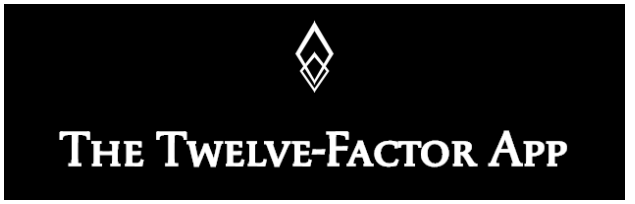
With cloud kettles,
you get the environment you need when you need it,
and throw it away when you are done.

B. Rederlechner

”

T · · Systems ·

THROUGHPUT LIMITED BY DEFECTS/ERROR RATE

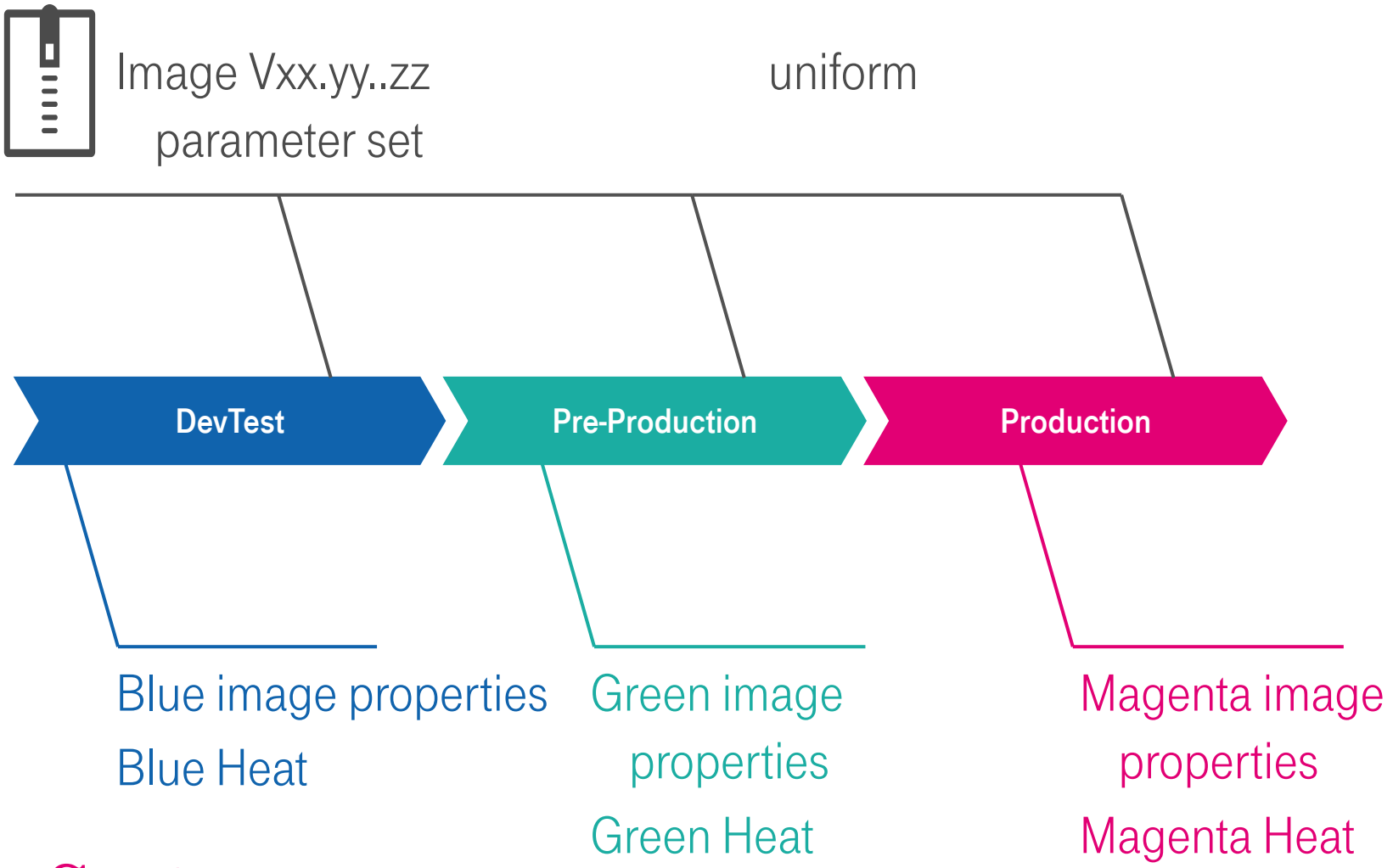


> **DEV/PROD PARITY:**
>
> Keep development, staging, and production as similar as possible.

T · · Systems ·

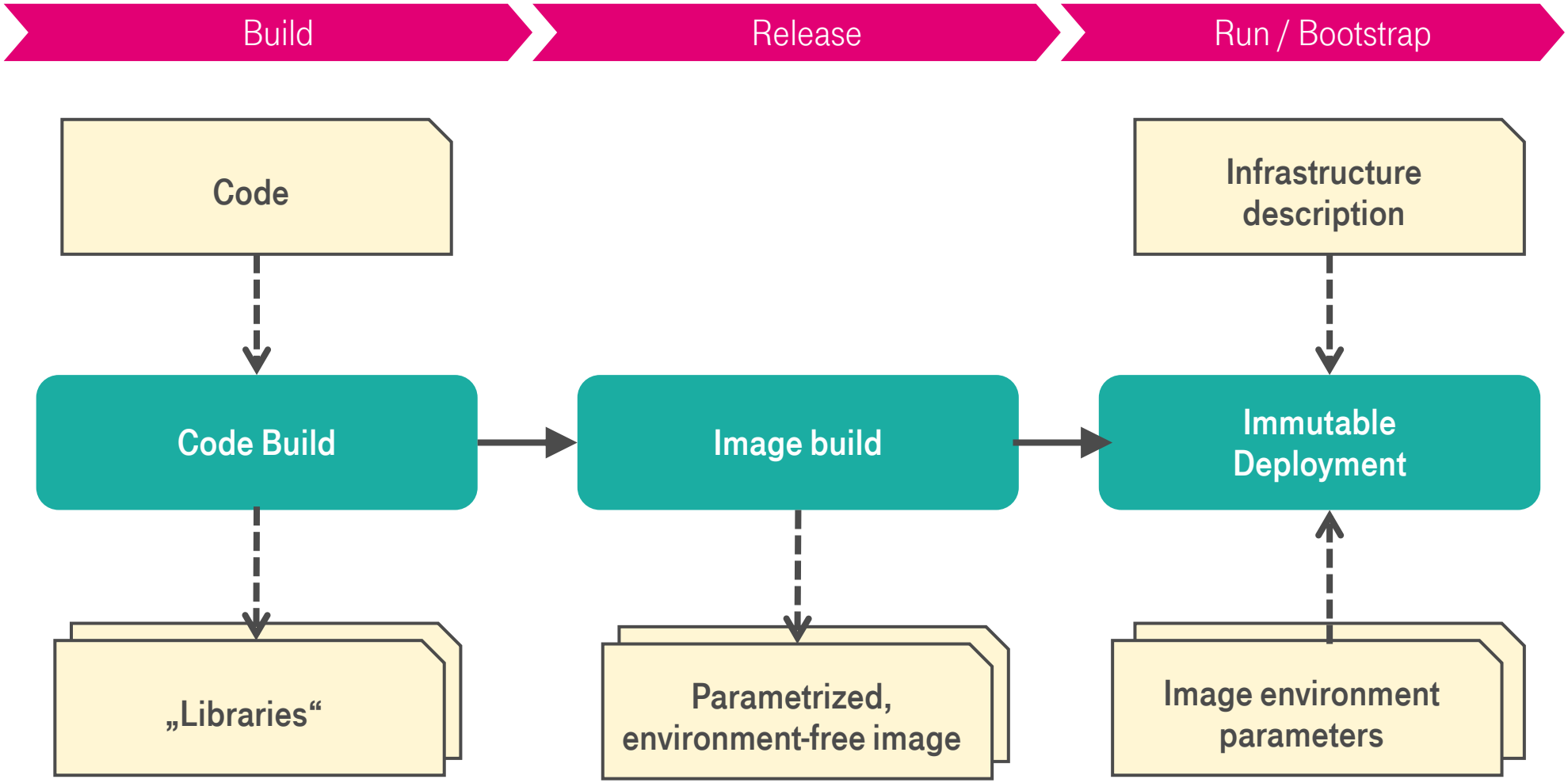
BUILD IMAGE ONCE – RUN ANYWHERE

MAXIMIZE DEV/PROD PARITY



T · · Systems ·

INFRASTRUCTURE AS CODE & IMMUTABLE INFRASTRUCTURES



T · · Systems ·

“

Deliver fast, fail fast.

Fowler, Gray, Shore

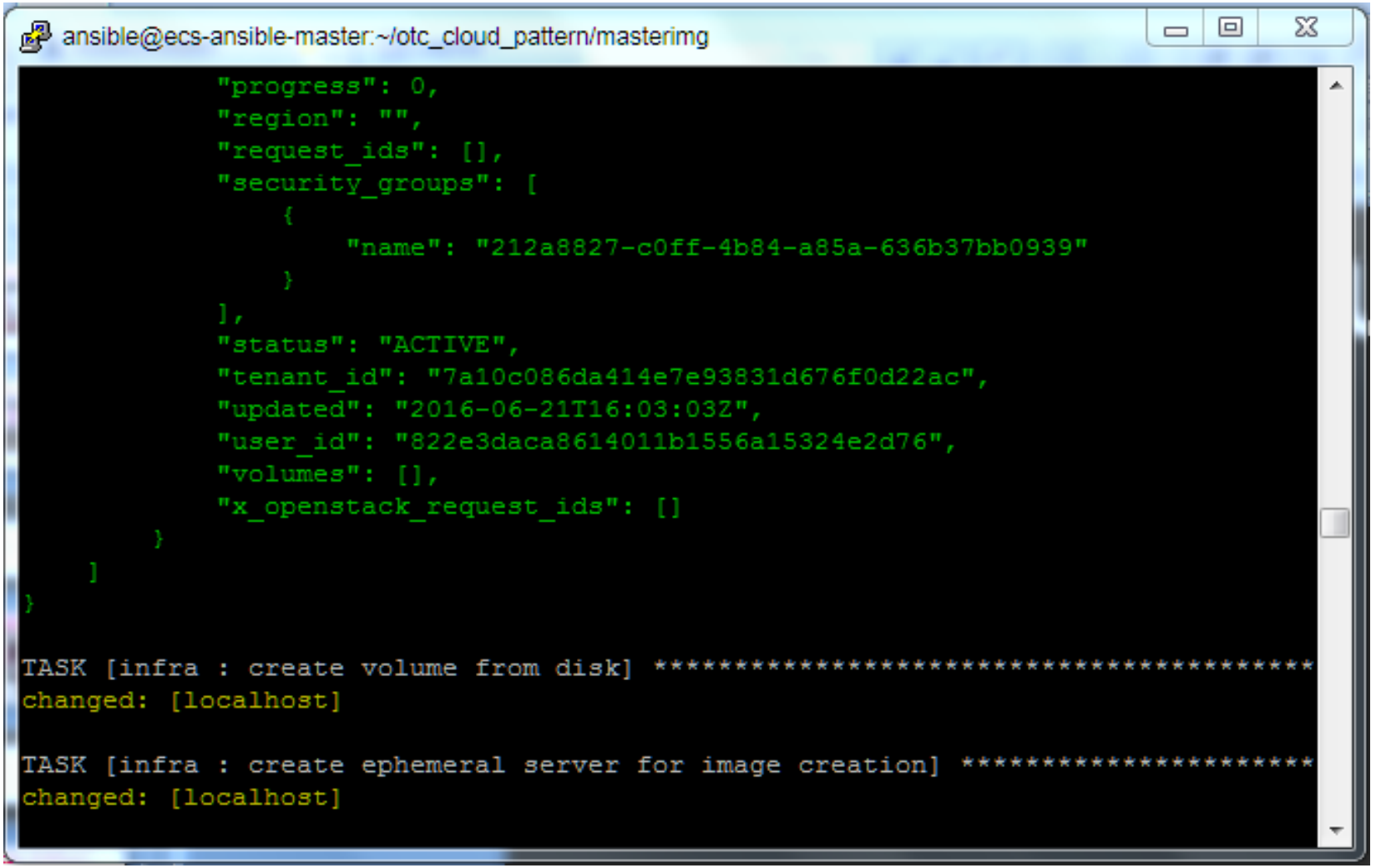
”

T · · Systems ·

EXERCISE/VIDEO 3: THE IMAGE BAKERY

IMAGE BAKERY

SEE ALSO „03_MASTERIMG“ EXERCISE



T · · Systems ·

ANTI-PATTERN FOR BAKED IMAGES AND EPHEMERAL SERVERS

No environment context within image:

Initialisation script in image:

Wrong:

```
if ( environment == `test` ) // enable some interface
```

Better: Functional switches

```
if ( interface_enabled == true ) // enable some interface
```

Reason: Setup decisions should not be hardcoded.

Dynamic, initial image configuration in a setup repository/DB:

Reason: If setup repository DB FAILS, NOTHING will boot

→ repository /DB becomes role „kernel“

→ must be ultra-available, but with hardcoded configuration

State-of-the-art: config by environment variables

AGENDA

01	Intro: Continuous Delivery Pipeline revisited
02	Reproducibility: Everything as Code
	02.1 Exercise: Trigger pipeline (Jenkins from GIT)
03	Provisioning time: Infrastructure as Code
	03.1 Exercise: Automated install and configuration
04	Dev-Prod-parity: Immutable Infrastructures
	04.1 Exercise/Video: Image bakery
05	**Conways law and continuous feedback**
	05.1 Exercise: ELK logging as microservices
06	Learn: Red/black, A/B and canaries
	06.1 Exercise: My little A/B test

PATH 1:
Optimize flow

PATH 2:
Include feedback

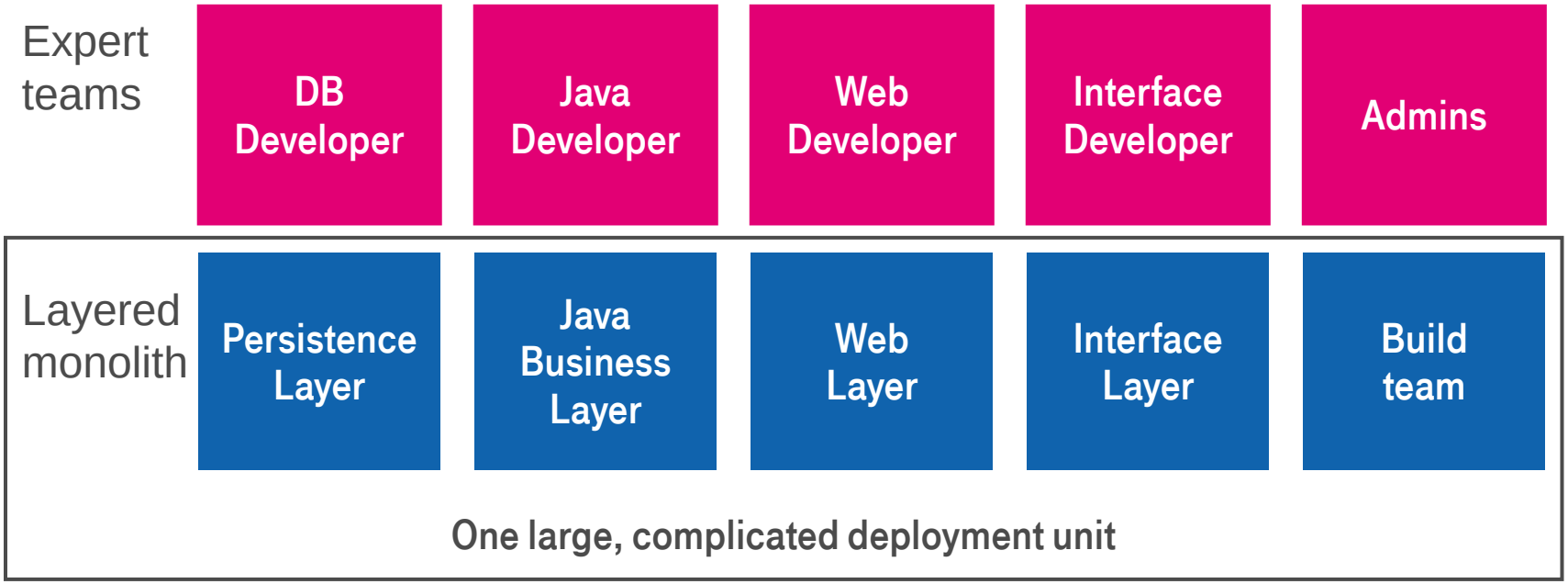
PATH 3:
Learn from system

T · · Systems ·

THROUGHPUT LIMITED BY #TEAMS LIMITED BY COST

CONWAYS LAW:
EXPERT TEAMS VS. CROSS FUNCTIONAL TEAMS

> Any organization that designs a system
> (defined more broadly here than just information systems)
> will inevitably produce a design whose structure is
> *a copy of the organization's communication structure.*



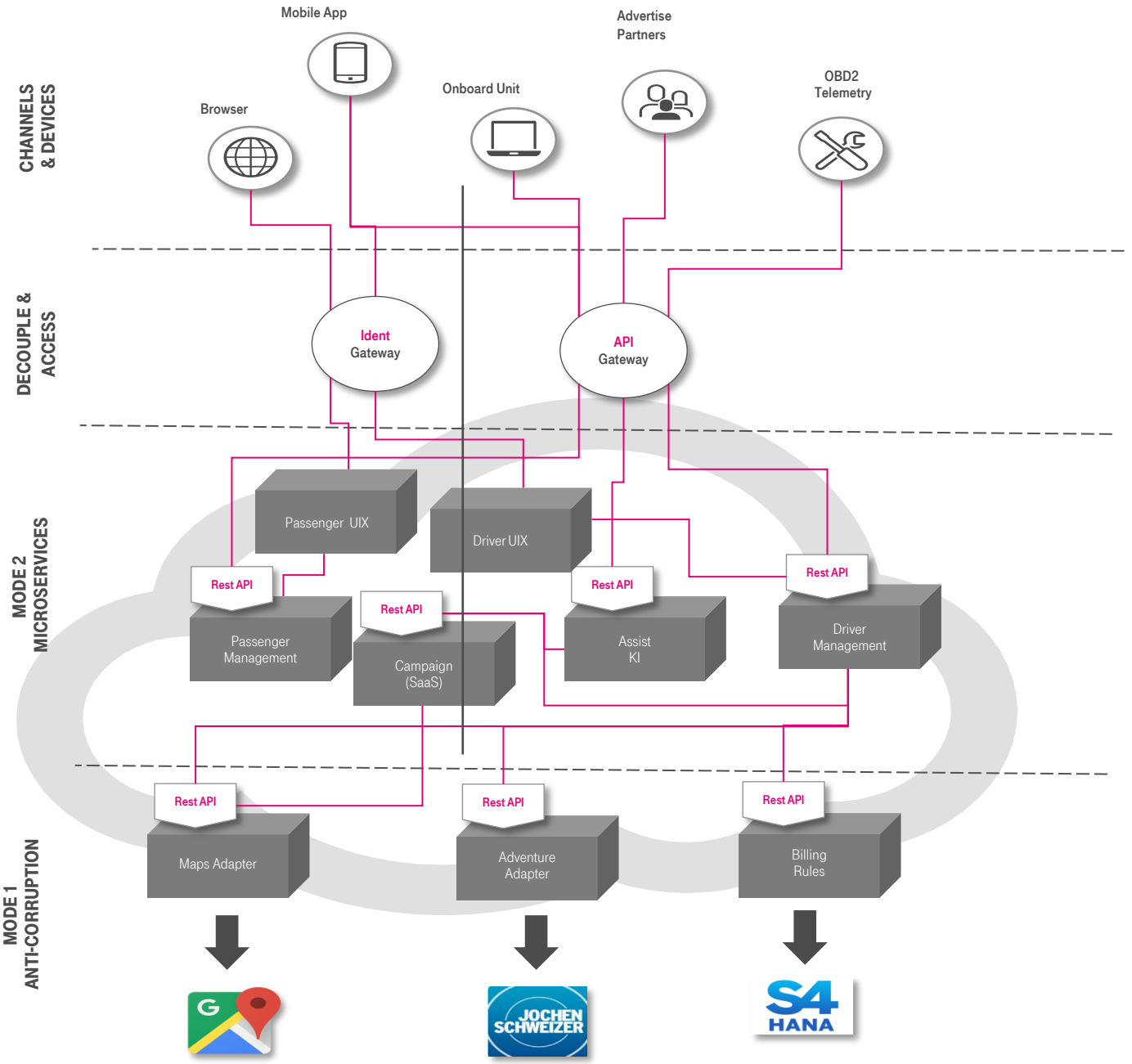
CROSS-FUNCTIONAL DEVOPS TEAMS & MICROSERVICES
ARCHITECTURE – ORGANISATION COMPATIBILITY

A Microservice is a well-defined part of a complex software system with the following properties:

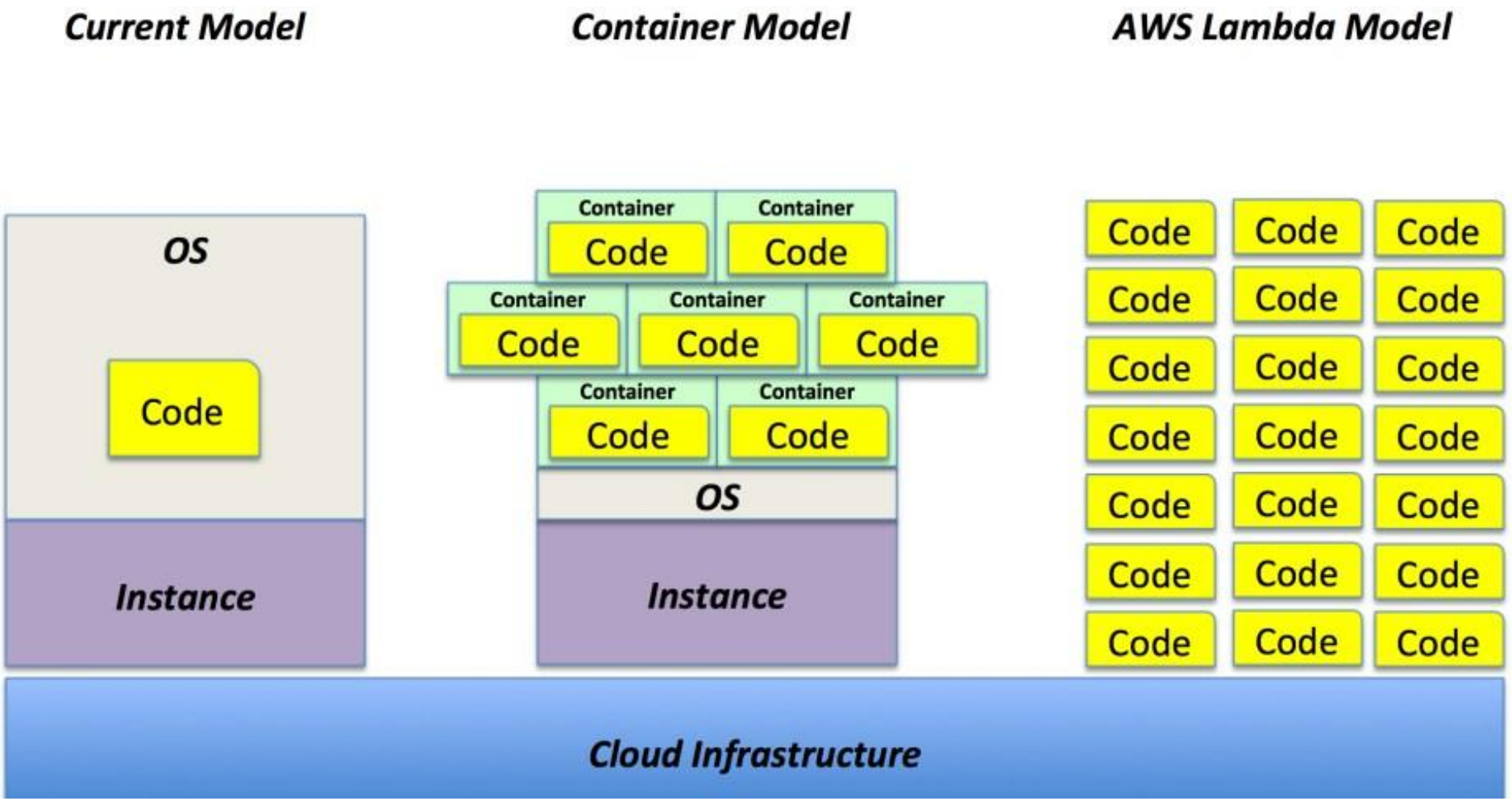
- It represents a dedicated, bounded-business context [DDD]
- It is an independent-deployable package.
- It delivers a defined service independent of its consumers and their bounded-contexts. [SOA]
- It is consumed only by a well-defined, formal API. [API]
- It is delivered and operated by an independent, cross-functional team (= „a team with all neccessary skills to deliver and run“) [DevOps]

MICROSERVICE ARCHITECTURE

Complexity
by
distributed system/
call dependencies
is the price to pay.



RESSOURCE USAGE & PACKAGE DENSITY



http://it20.info/2014/12/cloud-native-applications-for-dummies/

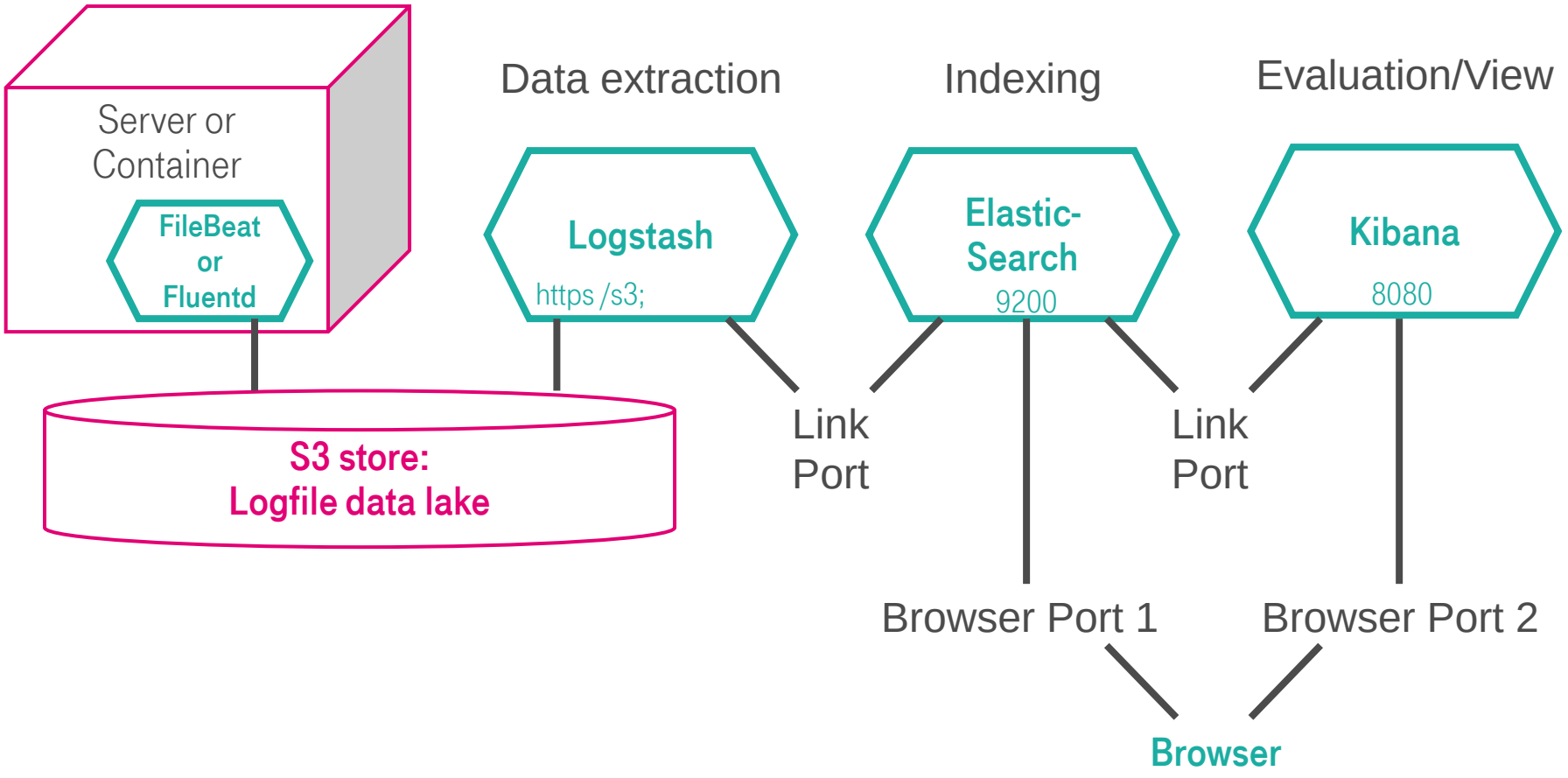
T · · Systems ·

EXERCISE 4: ELK LOGGING AS MICROSERVICES

HTTPS://BITBUCKET.ORG/GDCCLOUDCONSULTING/OTC_CCE_PATTERN

ELASTICSEARCH – LOGSTASH – KIBANA

MICROSERVICE EXAMPLE AND EVALUATION PLATFORM



STEP 1: BUILD DOCKER IMAGES

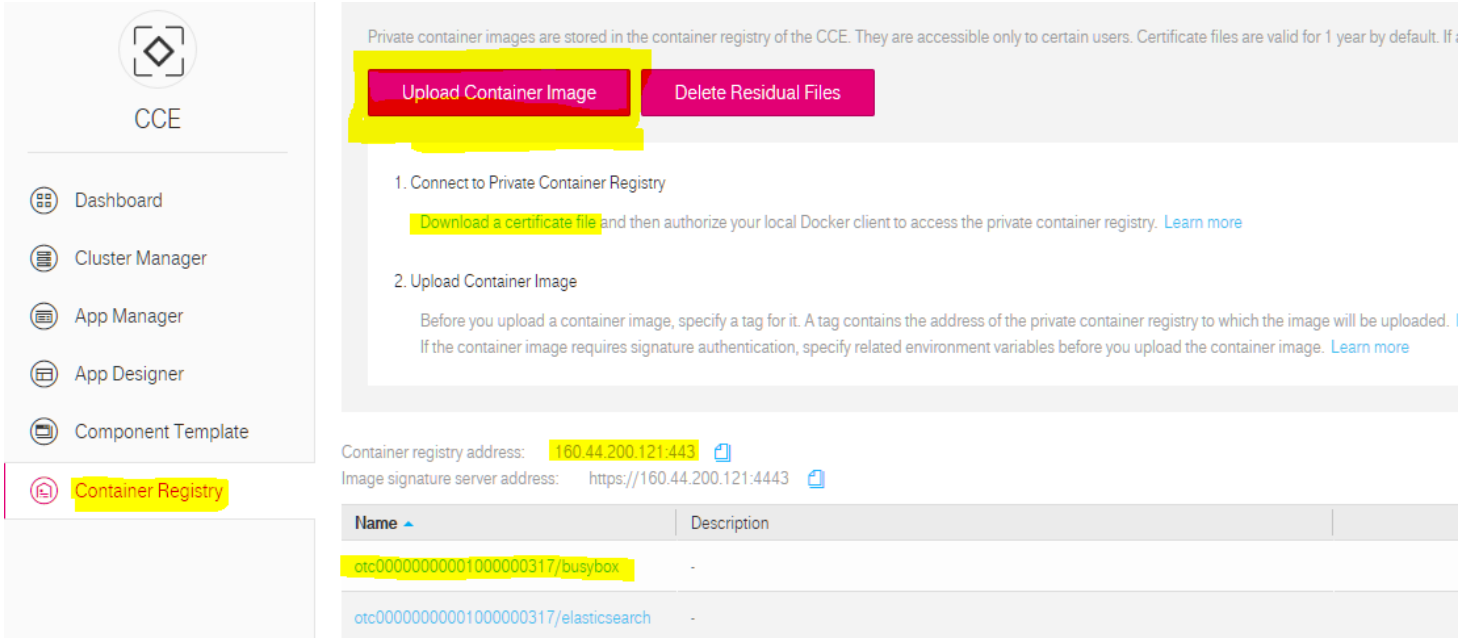
https://bitbucket.org/gdccloudconsulting/otc_cce_pattern/src/13f6d20ff5e9b1f94298c61ebc503aac7ddcc088/cce-master/?at=master

Download dockercfg.txt certificate from OTC CCE user interface, Container Registry. Copy the content to `~/.docker/config.json`.

Note: Remove any linebreaks in file, everything MUST be in one line!

Add the public IP address of your private registry to /etc/docker/daemon.json:

```
{ "insecure-registries": ["160.44.200.121:443"] }     -- already perpared for you!
```



T · · Systems ·

BUILDING THE FIRST IMAGE AND UPLOAD TO REGISTRY

Build base image and push

```
$ cd java-centos-base

$ docker build -t 160.44.200.121:443/otc-eu-de-00000000001000023731/otclabXX-java8-
centos7-base:1.8 .

$ docker push 160.44.200.121:443/otc-eu-de-00000000001000023731/otclabXX-java8-
centos7-base:1.8
```

SETUP KUBERNETES ACCESS, INSTALL SERVICES

```
$ cd exercises/04_elk_obs/bin
```

Adapt otc_env.sh, use your credentials for Open Telekom Cloud

```
$ . ./otc_env.sh
$ ./cce_kubeaddclusters
$ kubectl config set-context dop-cluster01-ctx --namespace=otclabXX
$ kubectl config set current-context dop-cluster01-ctx
```

Remember to adapt Namespace to „otclabXX“ and image name in yaml files.

Install secret:

```
$ echo {"160.xx.xxx.xxx:443":{"auth":"X2xxxxxxx ....xxx==","email":""}} | base64 -w 0
```

from $HOME/.docker/config.json, without ~~{"auths":~~{"160.xx.xxx.xxx:443":{"auth":"X2xxxxxxx ....xxx==","email":""}}~~}~~

```
$ vi otclabXX-pull-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: otclabXX-secret
data:
  .dockercfg: <paste base64 encoded secret here>
type: kubernetes.io/dockercfg

$ kubectl create -f ./my-pull-secret.yaml
```

KUBERNETES: INSTALL SERVICES

Set your own NodePort in dev/kub-kibana-service.yaml **(300XX for your otclab nr)**

```
$ cd elasticsearch
$ kubectl create -f dev/kub-elasticsearch-service.yaml
$ kubectl create -f dev/kub-elastictransport-service.yaml

$ cd kibana
$ kubectl create -f dev/kub-kibana-service.yaml
```

Review in Kubernetes web view:

```
Start your ssh with Tunnel option -L localhost:80XX:localhost:80xx
Do not use „127.0.0.1“ !

$ kubectl proxy --port=80XX &
```

Open your browser with localhost:80XX

INSTALL ELASTICSEARCH AND KIBANA

Remember to adapt imagename to your own images in Dockerfile:
FROM 160.44.200.121:443/otc-eu-de00000000001000023731/otclabXX-…

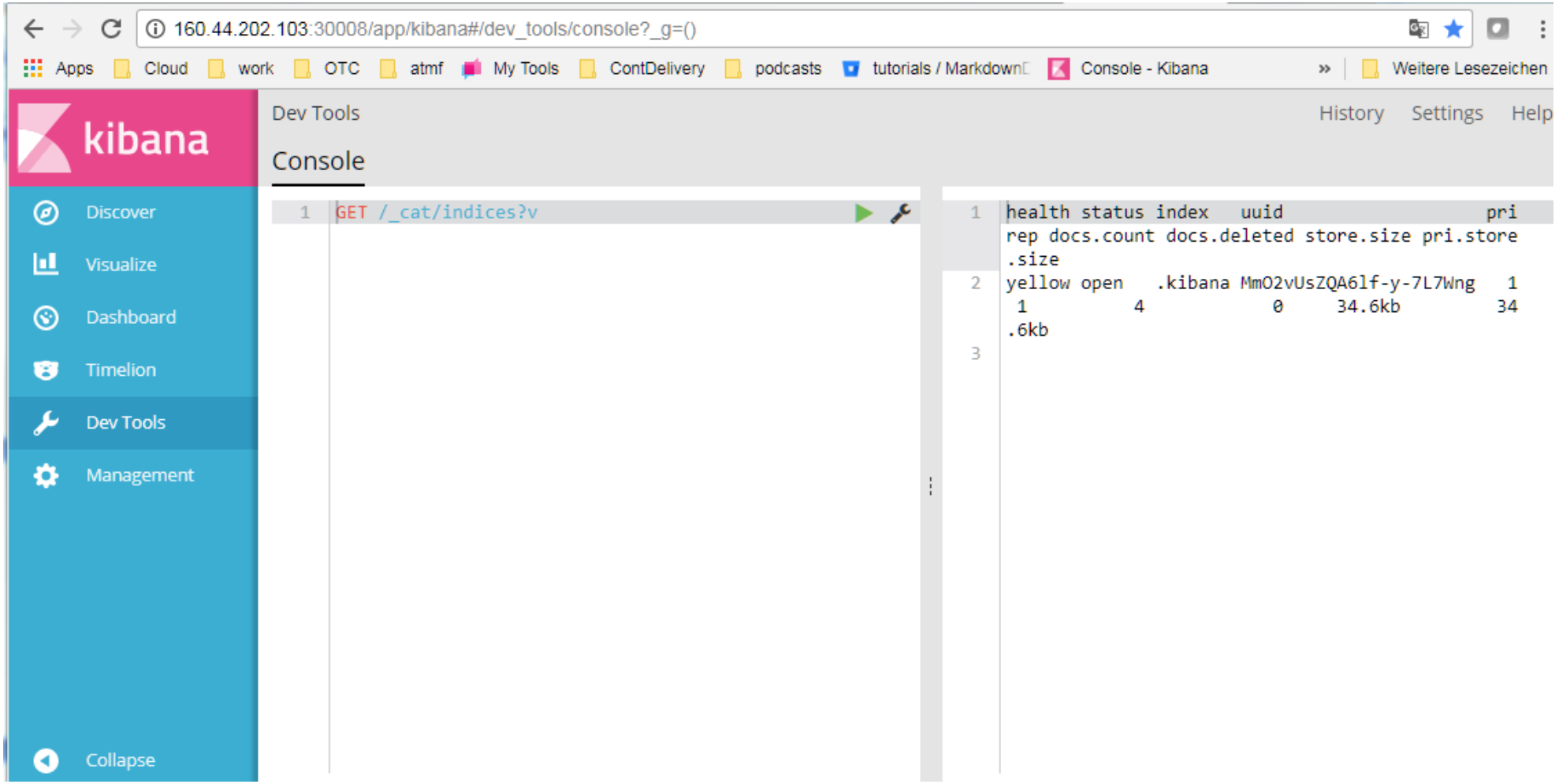
Elasticsearch:

```
$ cd elasticsearch

$ docker build -t 160.44.200.121:443/otc-eu-de-00000000001000023731/otclabXX-
elasticsearch:5.6.1 .
$ docker push 160.44.200.121:443/otc-eu-de-00000000001000023731/otclabXX-
elasticsearch:5.6.1

$ kubectl create –f dev/kub-elasticsearch-deploy.yaml
```

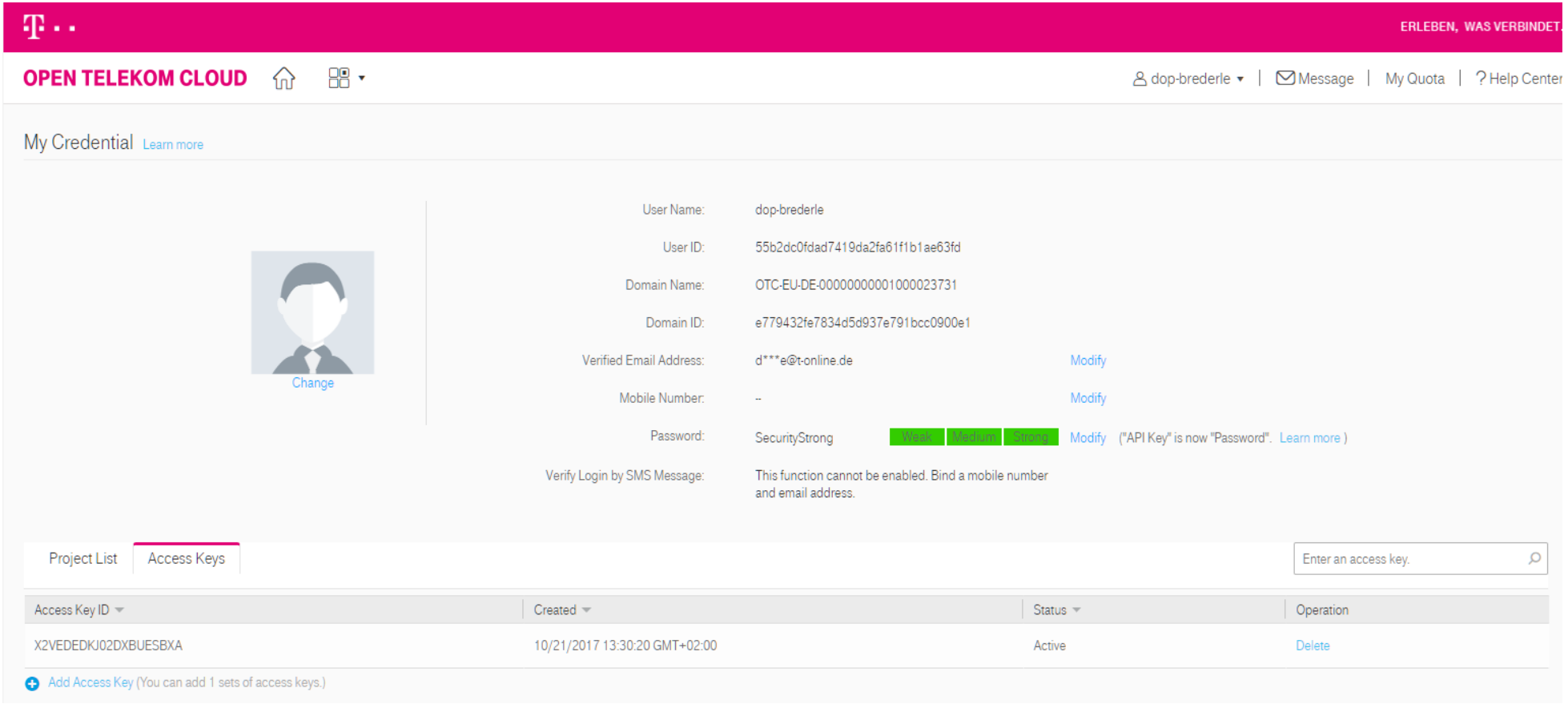
Kibana:

```
$ cd elasticsearch

$ docker build -t 160.44.200.121:443/otc-eu-de-00000000001000023731/otclabXX-
kibana:5.6.1 .
$ docker push 160.44.200.121:443/otc-eu-de00000000001000023731/otclabXX-kibana:5.6.1

$ kubectl create –f dev/kibana/kub-kibana-deploy.yaml
```

CHECK FOR RUNNING KIBANA



GET KEY FOR OBJECTSTORE



OPTIONAL: UPLOAD FILES TO OBJECTSTORE

```
$ s3cmd --configure

$ vi ~/.s3cfg
host_base = „https://obs.eu-de.otc.t-systems.com“

$ s3cmd --configure

$ s3cmd ls

$ s3cmd put -ssl logs.jsonl s3://my-bucket-name
```

https://docs.otc.t-systems.com/en-us/doc/pdf/20170829/20170829170403_50525.pdf

INSTALL LOGSTASH

Edit conf/*.conf:

```
input {
  s3 {
    access_key_id => „<my AK from OTC>"
    secret_access_key => „<my SK from OTC>"
    endpoint => "https://obs.eu-de.otc.t-systems.com"
    bucket => "85bblog"
…
```

Install logstash:

```
$ cd logstash

$ docker build -t 160.44.200.121:443/otc-eu-de-00000000001000023731/otclabXX-
logstash:5.6.1 .
$ docker push 160.44.200.121:443/otc-eu-de00000000001000023731/otclabXX-logstash:5.6.1

$ kubectl create –f dev/kibana/kub-logstash-deploy.yaml
```

TIP: KUBERNETES DEBUG QUICKIES

Quick start of own Docker image in registry:

```
kubectl run -ti myjavabox --image=160.44.200.121:443/otc00000000001000000317/java-
centos7-base:0.1 -- bash
```

Quick start of a bare client linux box within cluster (e.g. to test IP connections)

```
kubectl run -ti mybox --image=busybox -- sh
```

Connect to a running pod:

```
kubectl exec -ti <pod-id> -- bash
```

Kubernetes cheat sheet:
https://kubernetes.io/docs/user-guide/kubectl-cheatsheet/

T · · Systems ·


KAFFEE-
PAUSE

AGENDA

01	Intro: Continuous Delivery Pipeline revisited
02	Reproducibility: Everything as Code
	02.1 Exercise: Trigger pipeline (Jenkins from GIT)
03	Provisioning time: Infrastructure as Code
	03.1 Exercise: Automated install and configuration
04	Dev-Prod-parity: Immutable Infrastructures
	04.1 Exercise/Video: Image bakery
05	Conways law and continuous feedback
	05.1 Exercise: ELK logging as microservices
06	**Learn: Red/black, A/B and canaries**
	06.1 Exercise: My little A/B test

PATH 1:
Optimize flow

PATH 2:
Include feedback

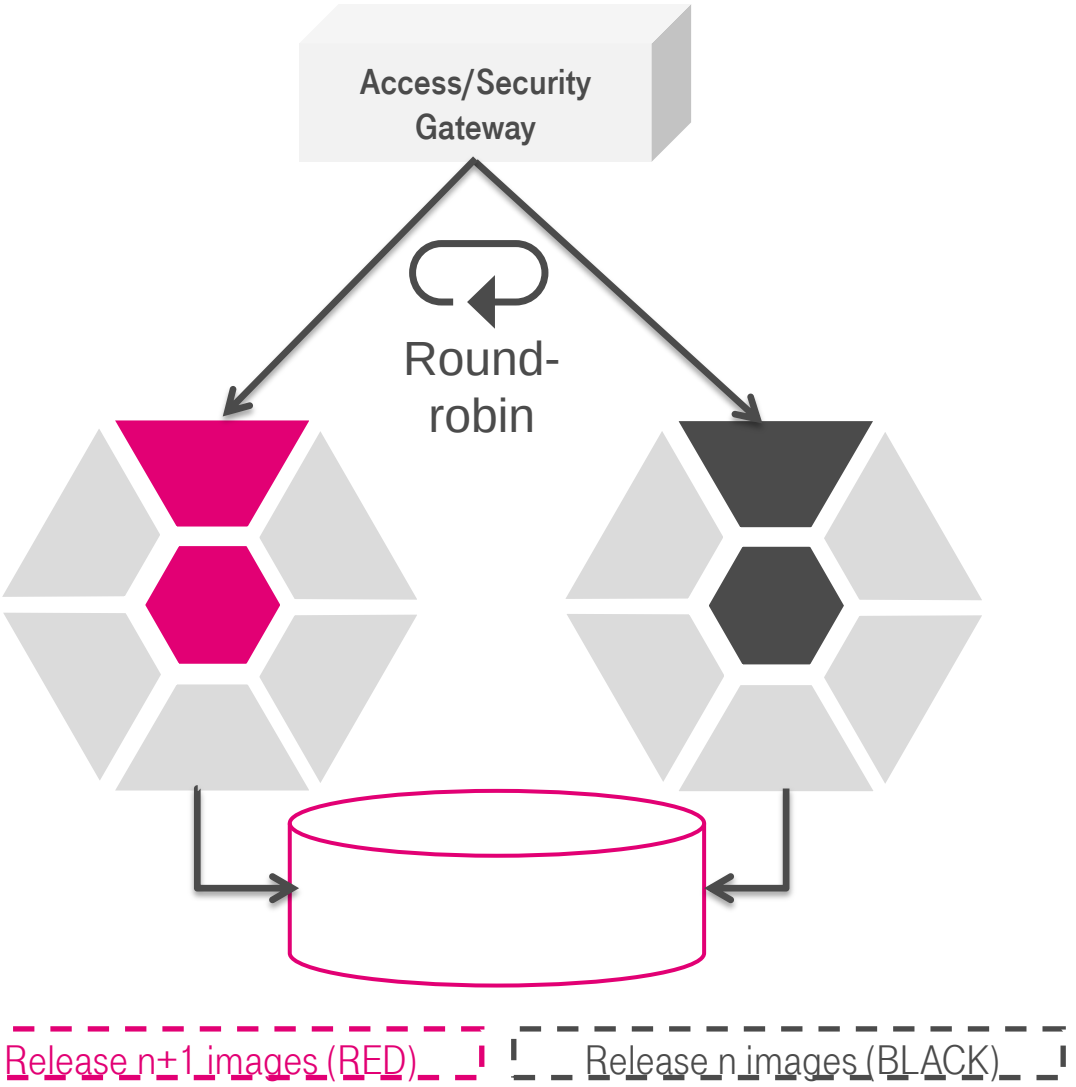
PATH 3:
Learn from system

T · · Systems ·

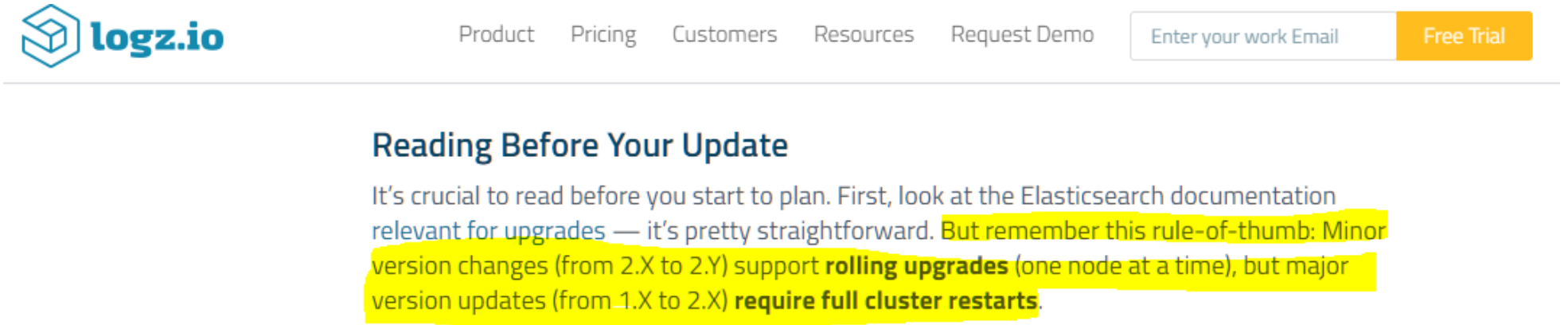
LOGGING REVISITED

LOG LEVEL	CONTENT	MAIN PURPOSE
`Fatal, Error, Warning`	Exceptions Unexpected behaviour	Alarming [prod enabled]
`INFO`	**Business events** Business correlation ID	**Business Learning** **[prod enabled]**
`DEBUG, TRACE`	Developer details	Debugging [prod disabled]

ROLLING UPDATE, RED/BLACK DEPLOYMENT

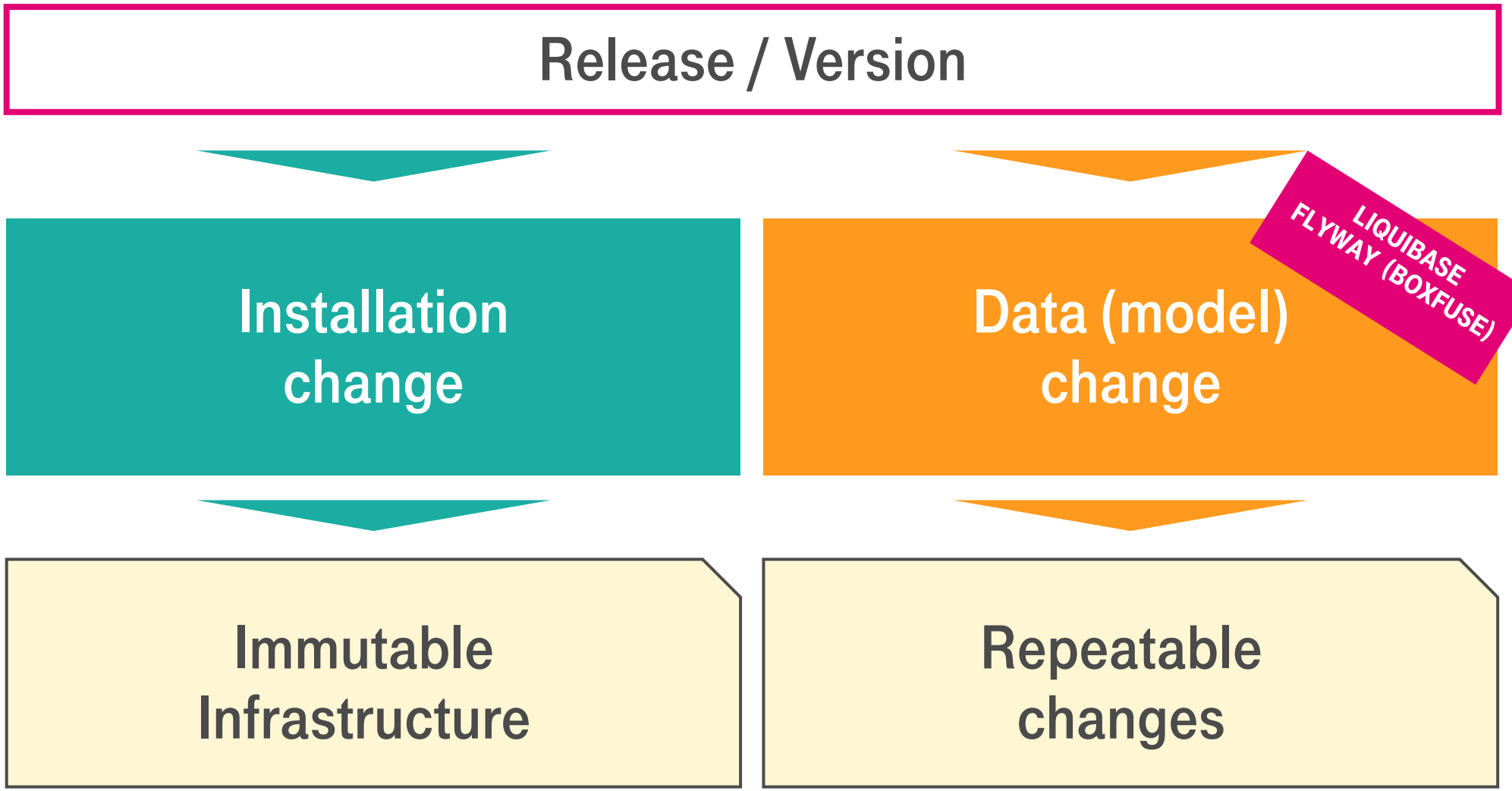


TIP: VERSIONING AND ROLLING UPDATES



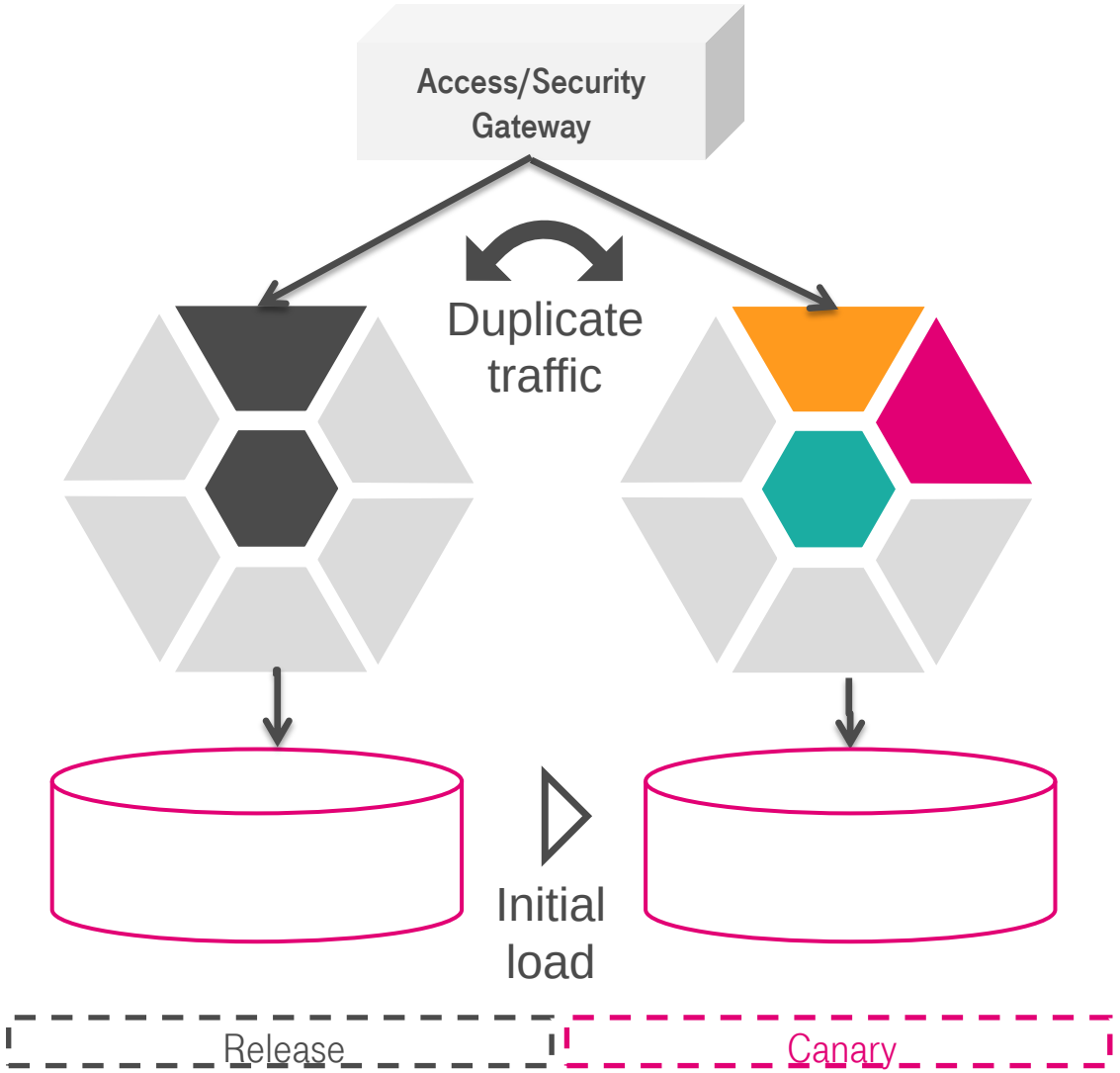
T · · Systems ·

RELEASES: DON´T FORGET YOUR DATA CHANGES!



T · · Systems ·

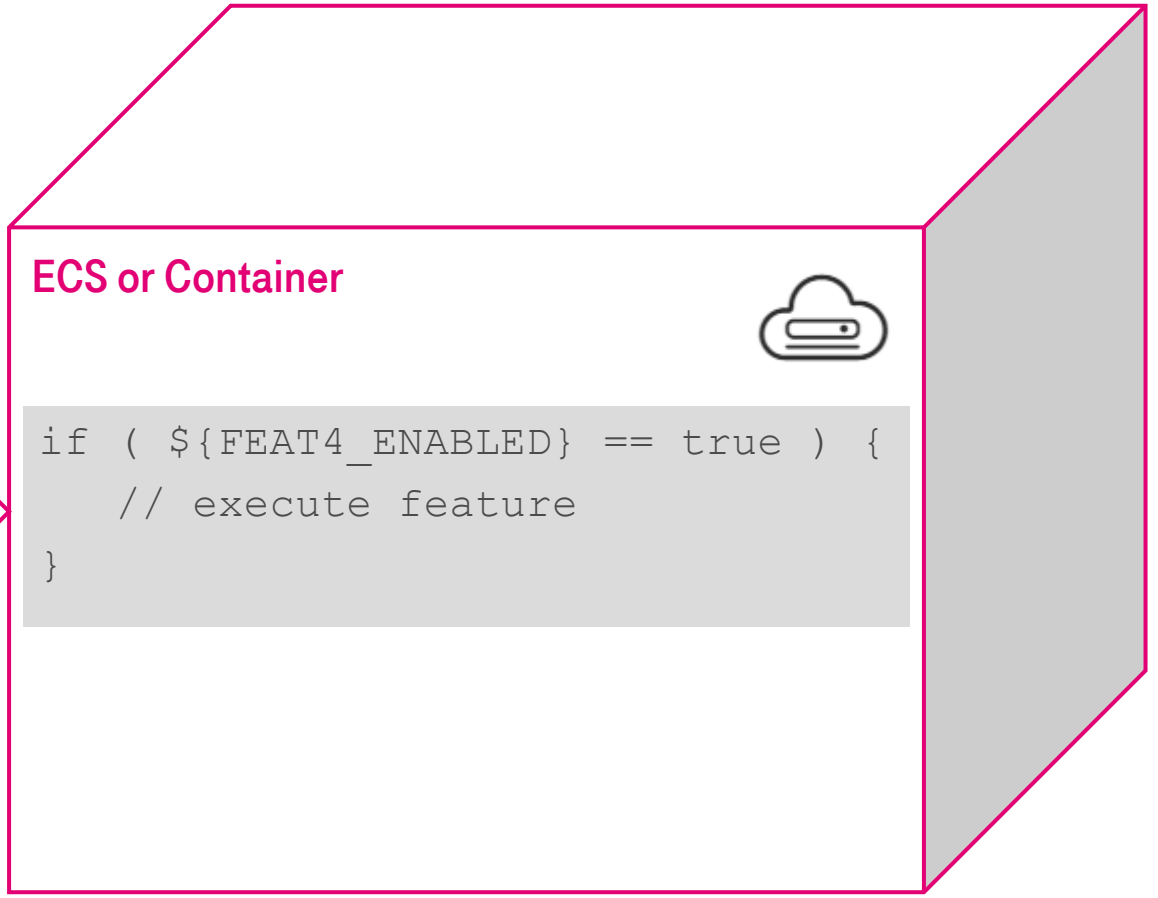
CANARY RELEASES



FEATURE TOGGLES (IN PRODUCTION)

```
export FEAT1_ENABLED=true
export FEAT2_ENABLED=false
export FEAT3_ENABLED=true
export FEAT4_ENABLED=true
export FEAT5_ENABLED=false
export FEAT6_ENABLED=true
export FEAT7_ENABLED=false
export FEAT8_ENABLED=false
…
```

Inject



A/B TESTS IN PRODUCTION

```
export FEAT1_RATIO=0.6
export FEAT2_RATIO=0.3
export FEAT3_RATIO=0.0
export FEAT4_RATIO=0.4
export FEAT5_RATIO=0.0
export FEAT6_RATIO=0.5
export FEAT7_RATIO=1.0
export FEAT8_RATIO=0.5
…
```

Inject

ECS or Container

```
if (random() > ${FEAT4_RATION}) {
  log(`FEAT4,variant A applied`)
  // execute old feature
} else {
  log(`FEAT4,variant B applied`)
  // execute new feature
}
```

6.1 LEARN:
MY LITTLE A/B TEST
(A FREE EXERCISE)

THANK YOU!

BACKUP

Partially done work!

Extra features,

Unused features!

Relearning!

TRANSPORTATION

Handoffs!

T · · Systems ·

Delays!

Task switching!

Defects!

THE SEVEN WASTES OF SOFTWARE DEVELOPMENT

#1 - Partially Done Work (Inventory)

#2 - Extra Features, unused features (Overproduction)

#3 – Relearning (Extra processing)

#4 – Handoffs (Transportation)

#5 – Delays (Waiting)

#6 - Task Switching (Motion, Jumping)

#7 - Defects