Jenkins Techlab

AGENDA

10 min.

Intro

Welcome, purpose of the lab, agenda

1 hour

Talk

Introduction to CI/CD and Jenkins Pipelines

~ 3 hours

Labs

Check Lab Setup,
Basic Labs,
Wrap up

30 min.

Recap

Recap basic labs

3.5 hours

Labs

Advanced Labs,
Q&A
Wrap up

TECHLAB OBJECTIVES

Achieve theory base

Learn basic Jenkins Pipeline features

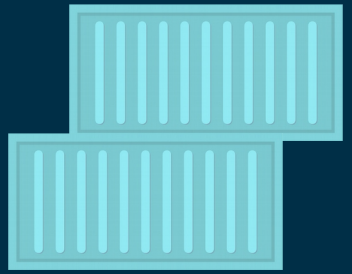
Learn advanced Jenkins Pipeline use-cases

CI/CD

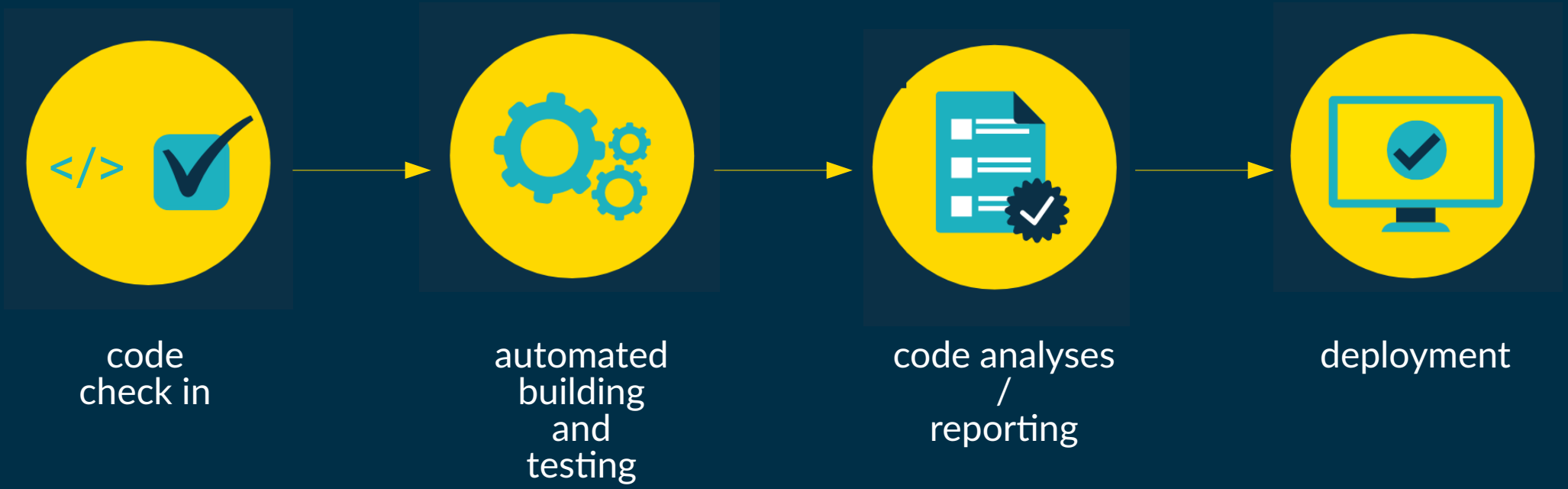
Once upon a time...

- «Integration Hell» und «Works on My Machine»
- 1991: Object Oriented Design: With Applications, Grady Booch
- 1997: Extreme Programming Explained

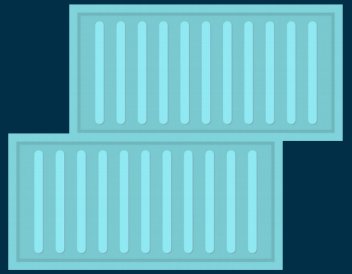
1



Continuous Integration


code
check in
automated
building
and
testing
code analyses
/
reporting
deployment

2



Continuous Delivery

«It aims at building, testing, and
releasing software faster and more frequently.»

Continuous Integration

fast & often released

on Click

Development process

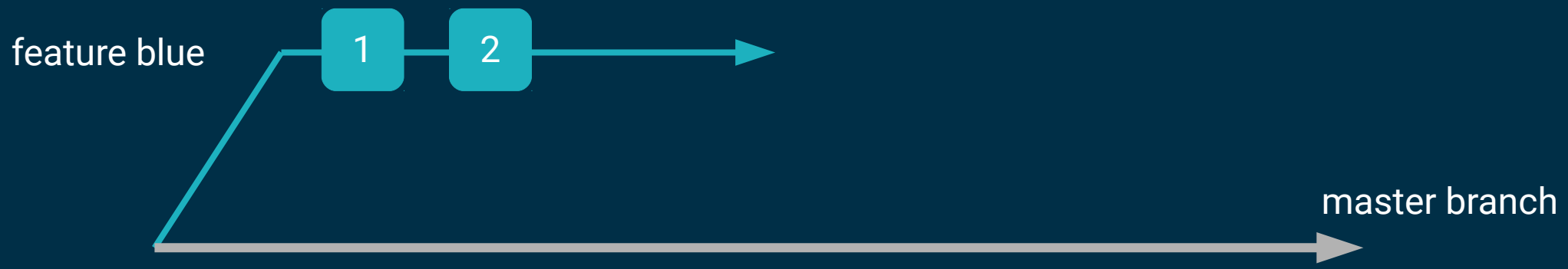
- Small and short development cycles
- Agile approach
- Fully automated acceptance tests
- Very high test coverage
- Feature toggles
- avoid “Big Scary Merge”

Feature Toggles

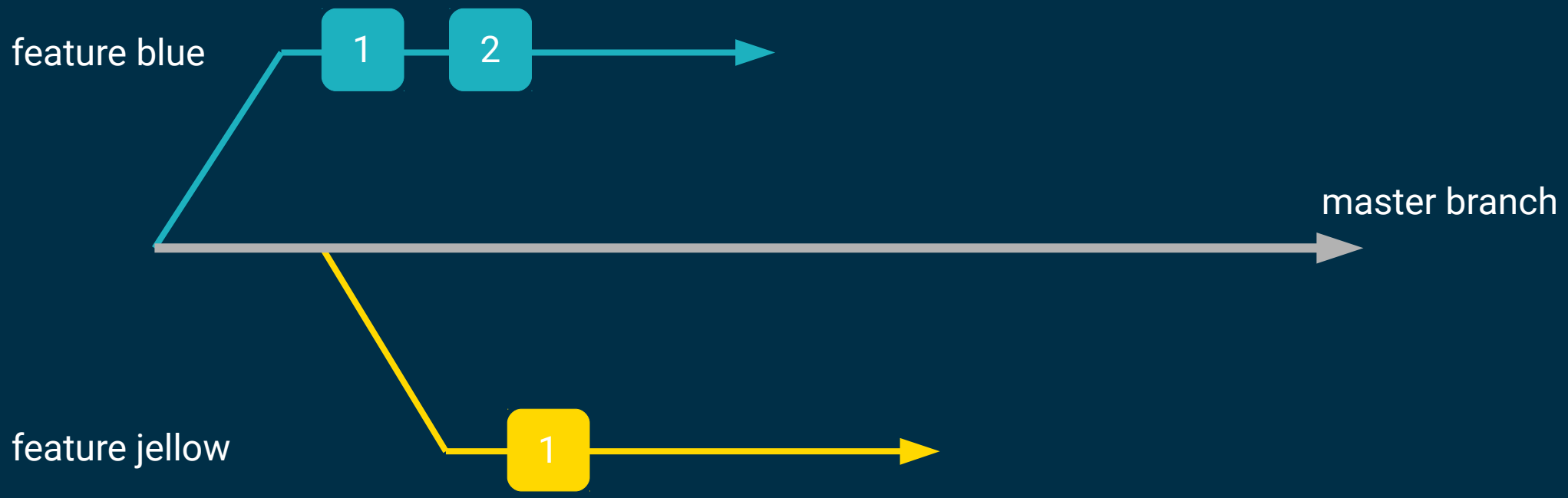
```
...

public void sendOrderNotification(Order order) {

    // send default OrderNotification
    mailService.sendOrderNotificationMessage(order);

    if(featureService.isEnabled(SEND_ORDER_NOTIFICATION_AS_SMS)){
        smsService.sendOrderNotificationMessage(order);
    }
}
...
```

Big Scary Merge of Feature Branches

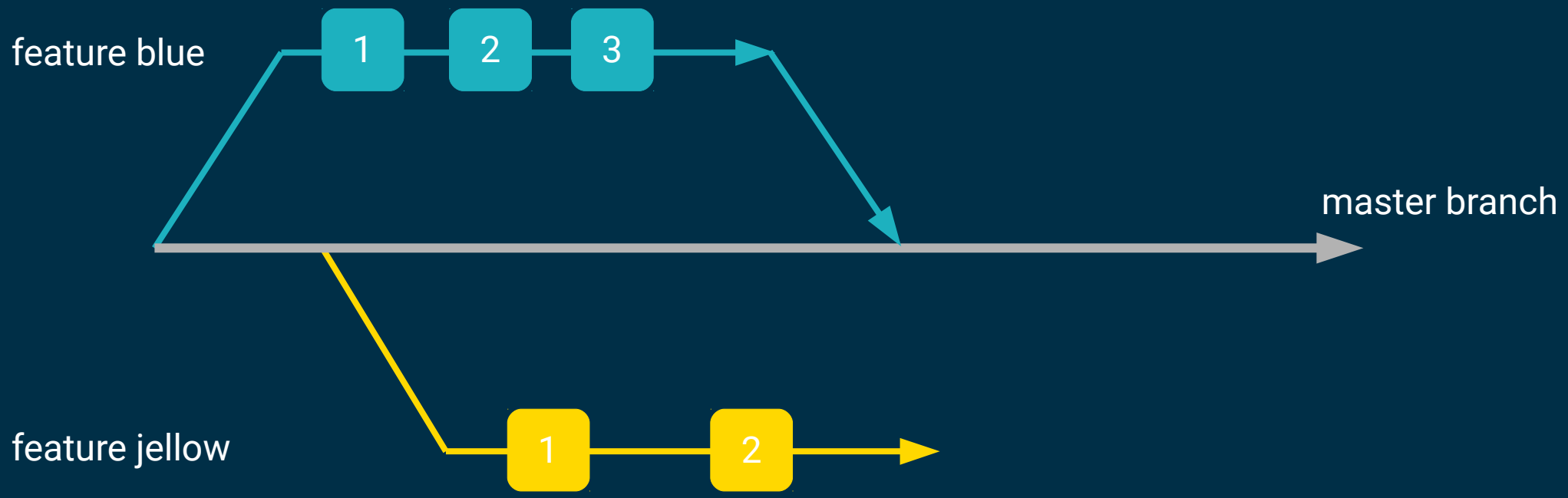
Big Scary Merge of feature branches



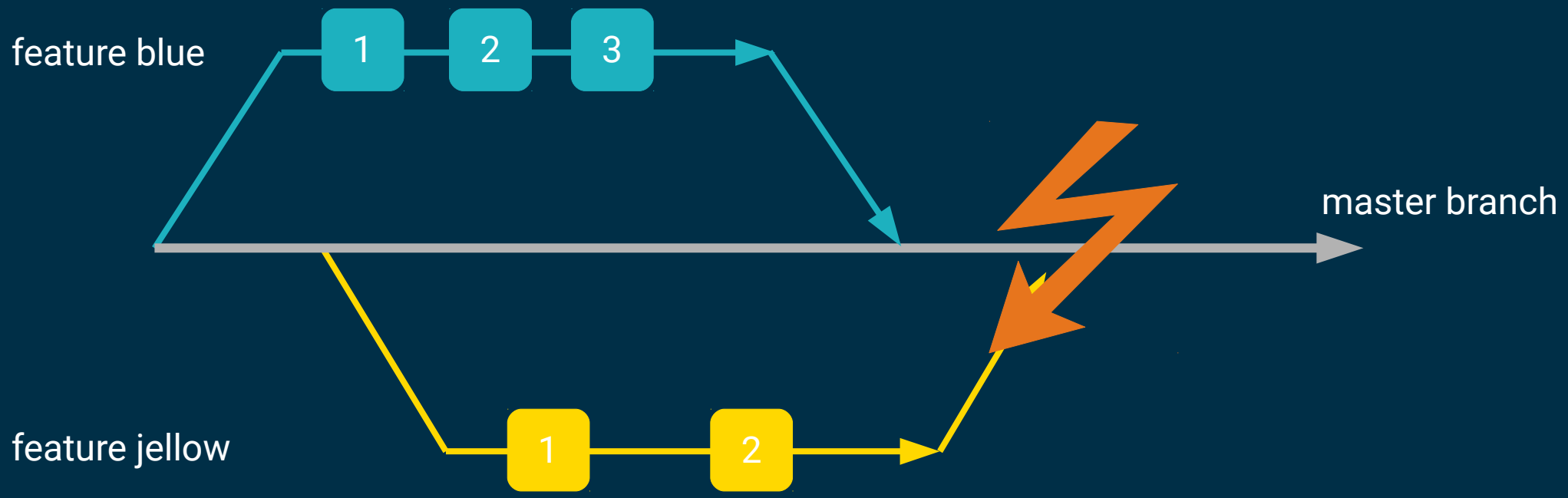
Big Scary Merge of feature branches



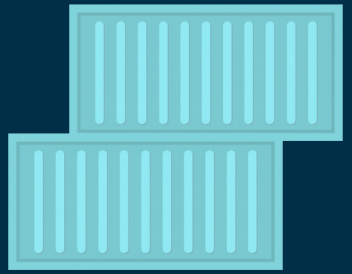
Big Scary Merge of feature branches



Big Scary Merge of feature branches



3



Continuous Deployment

Continuous Deployment

Each release is automatically deployed in production

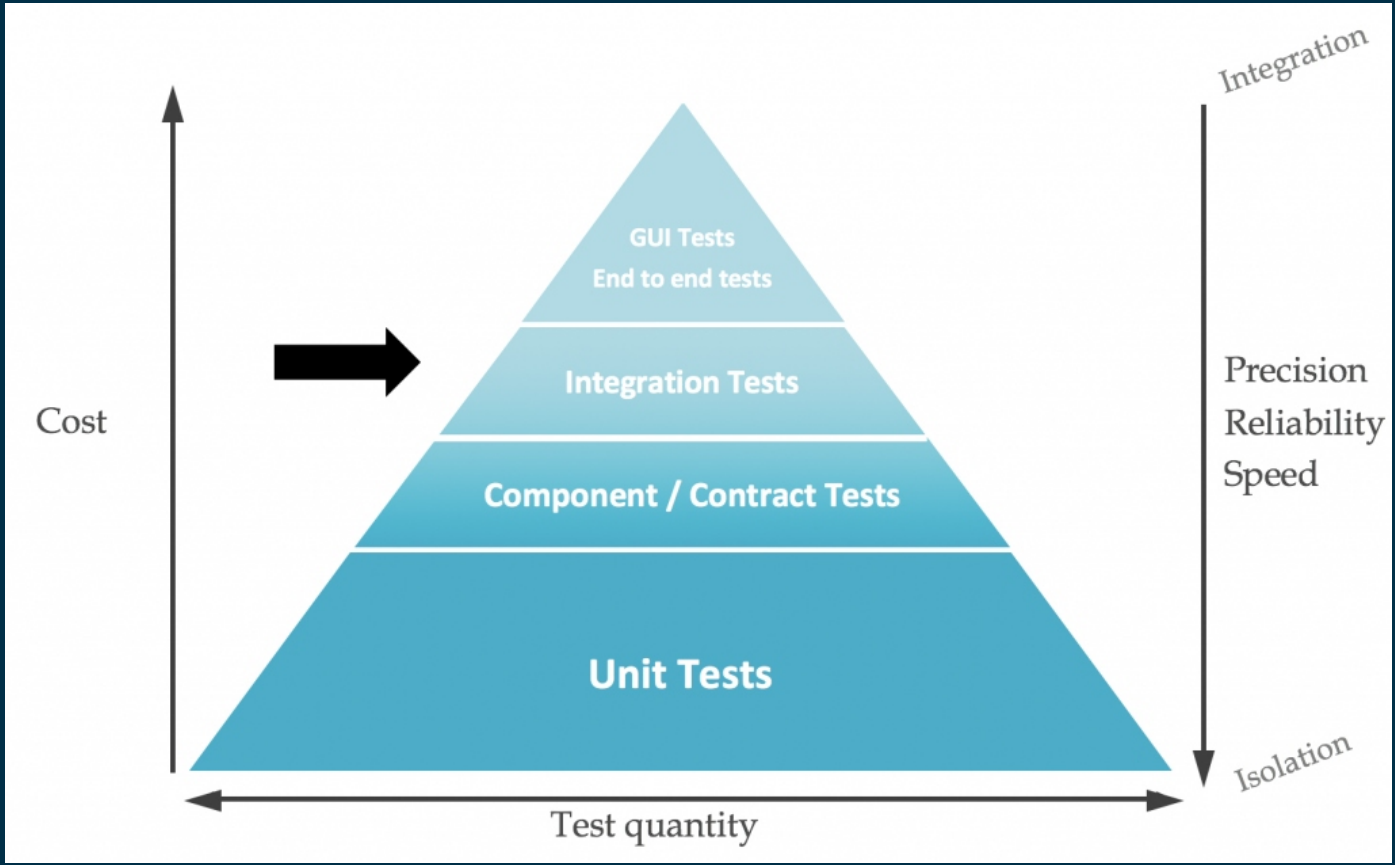
High test coverage needed

Uninterrupted deployments required

Blue – Green (Yellow) deployments

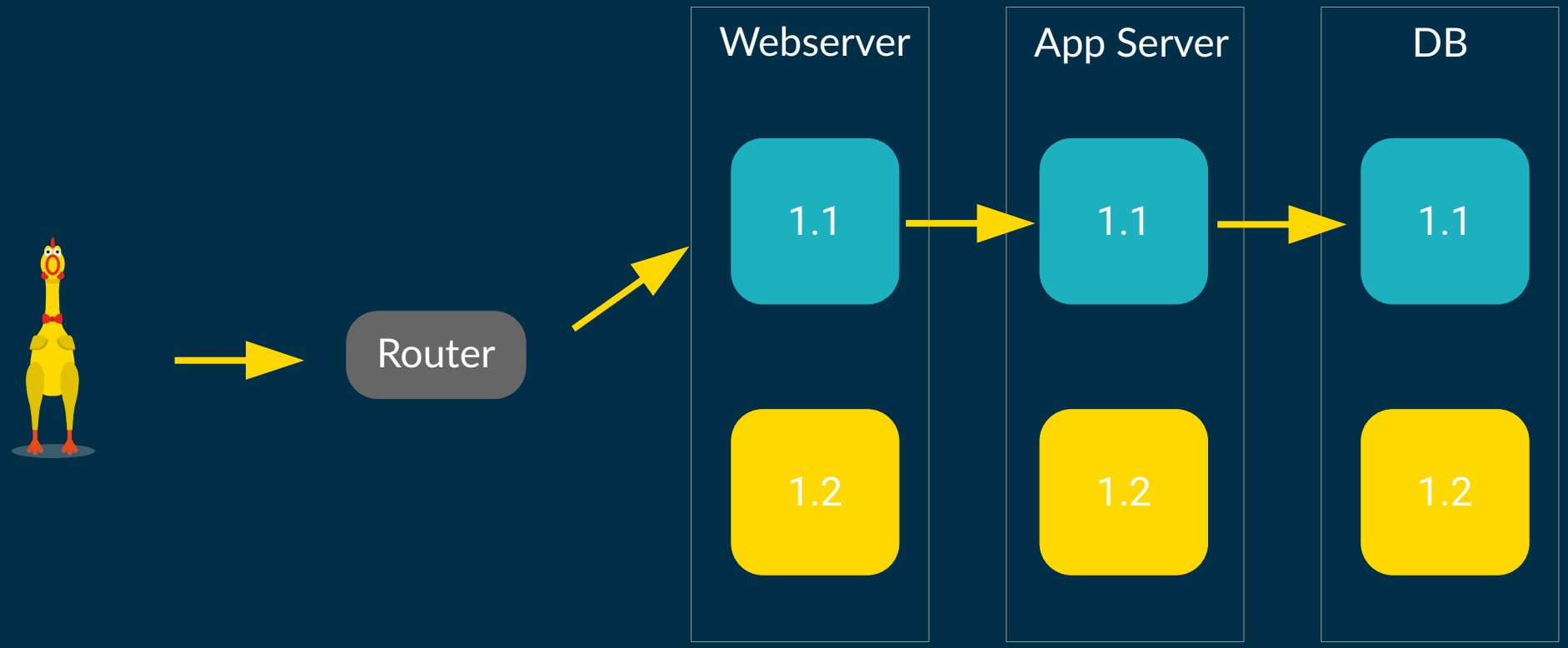
Canary releasing

Testing

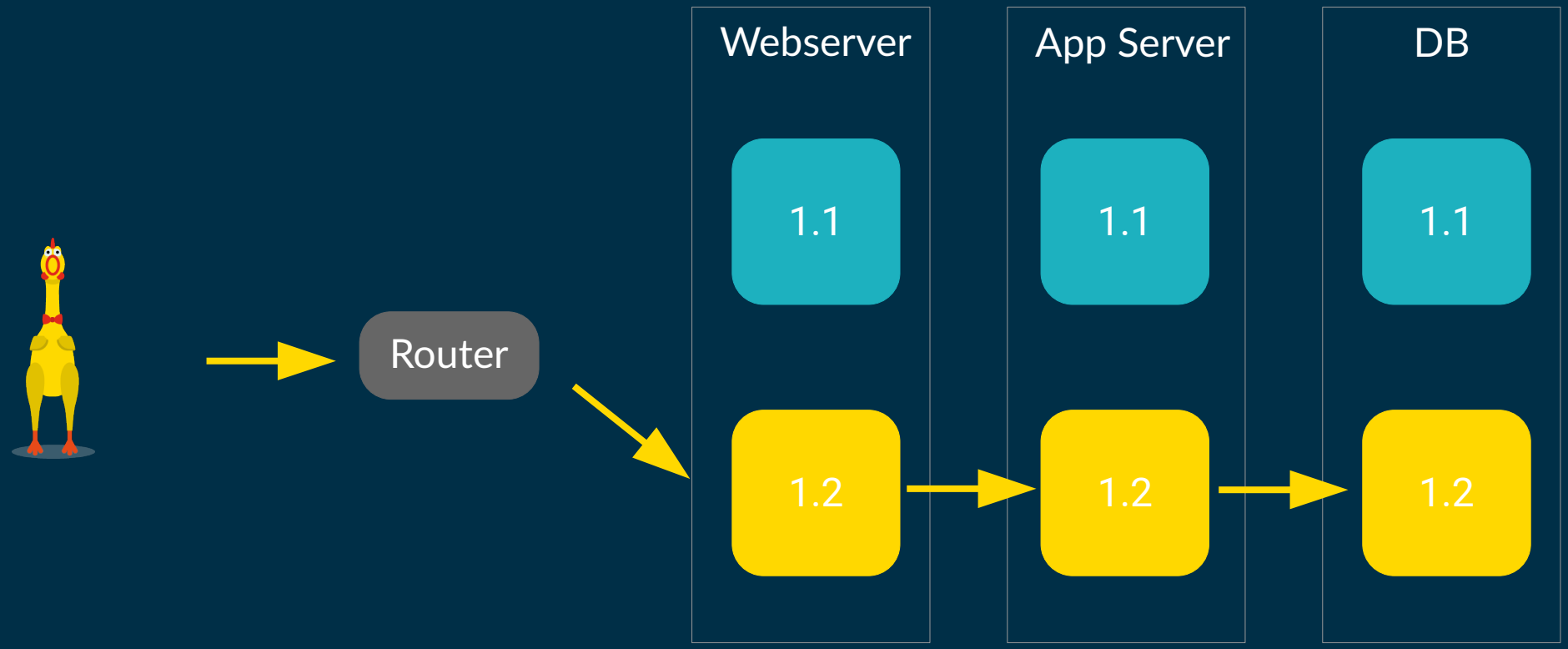


A good test concept is essential for Continuous Deployment.

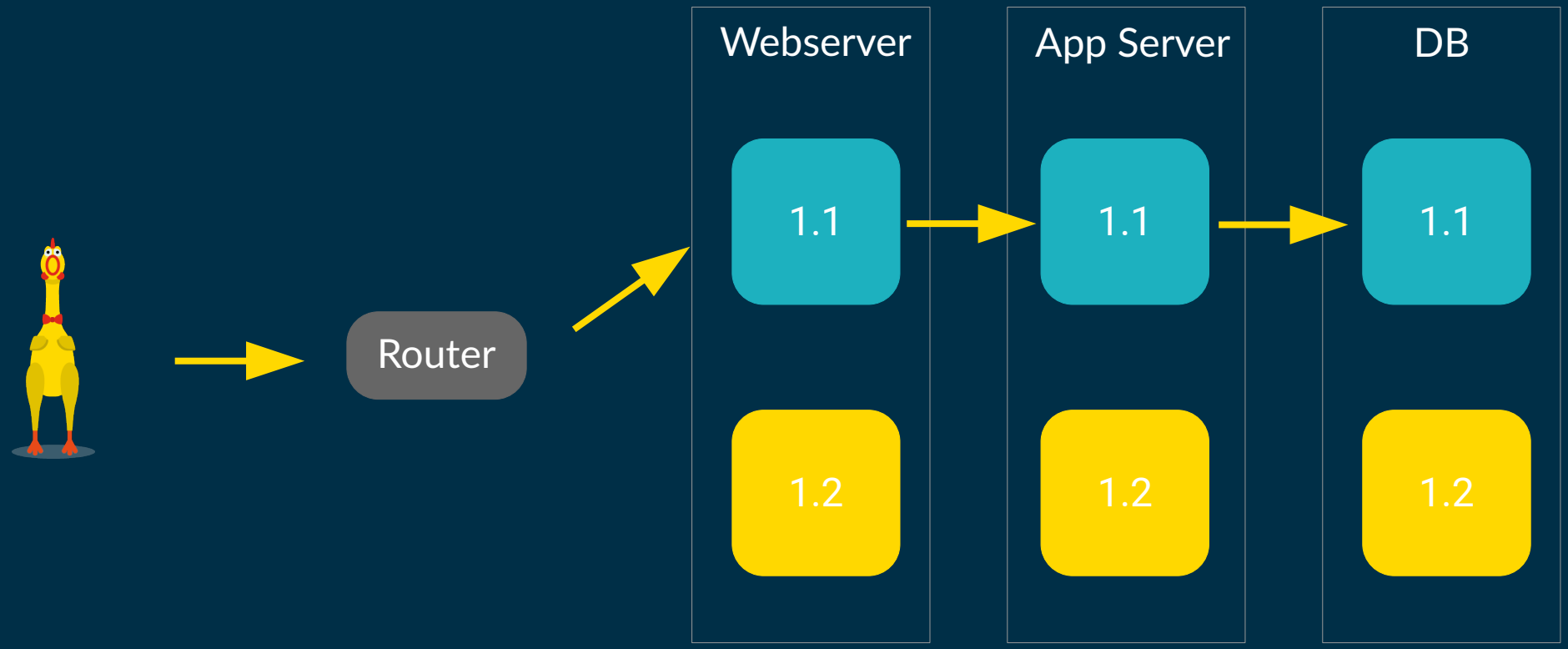
Blue - Yellow Deployments



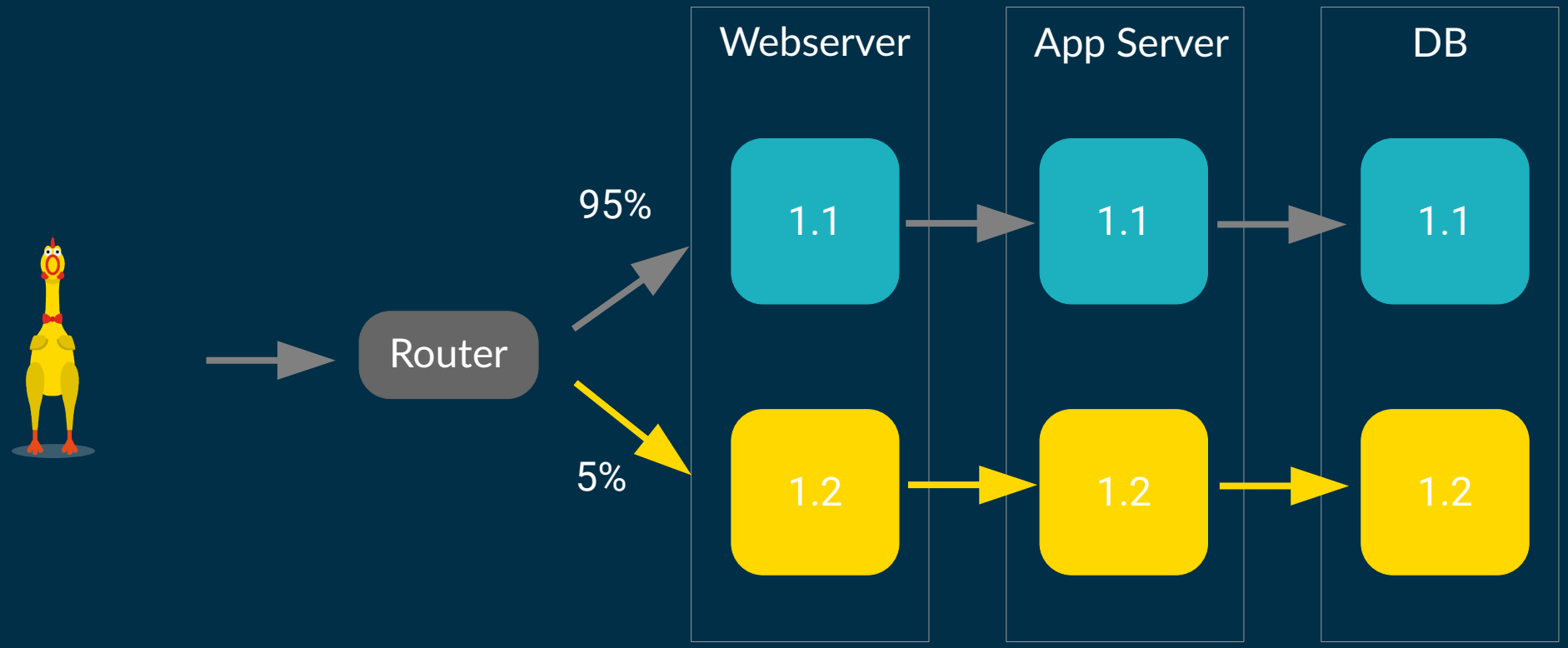
Blue - Yellow Deployments



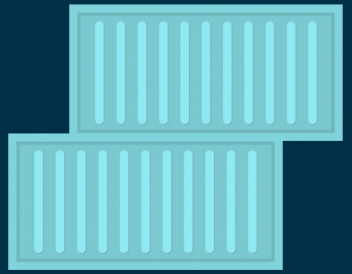
Canary Releasing



Canary Releasing

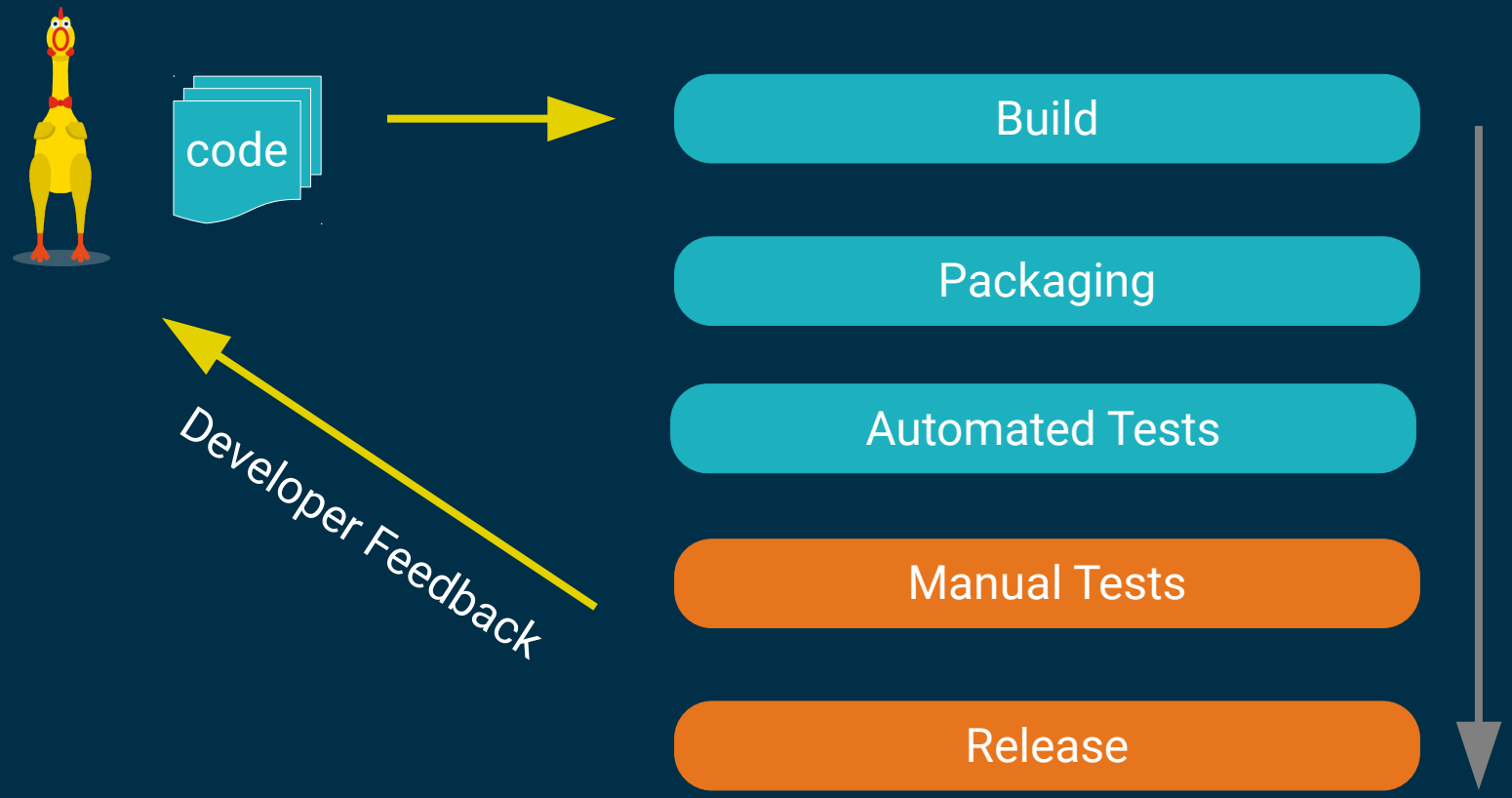


4

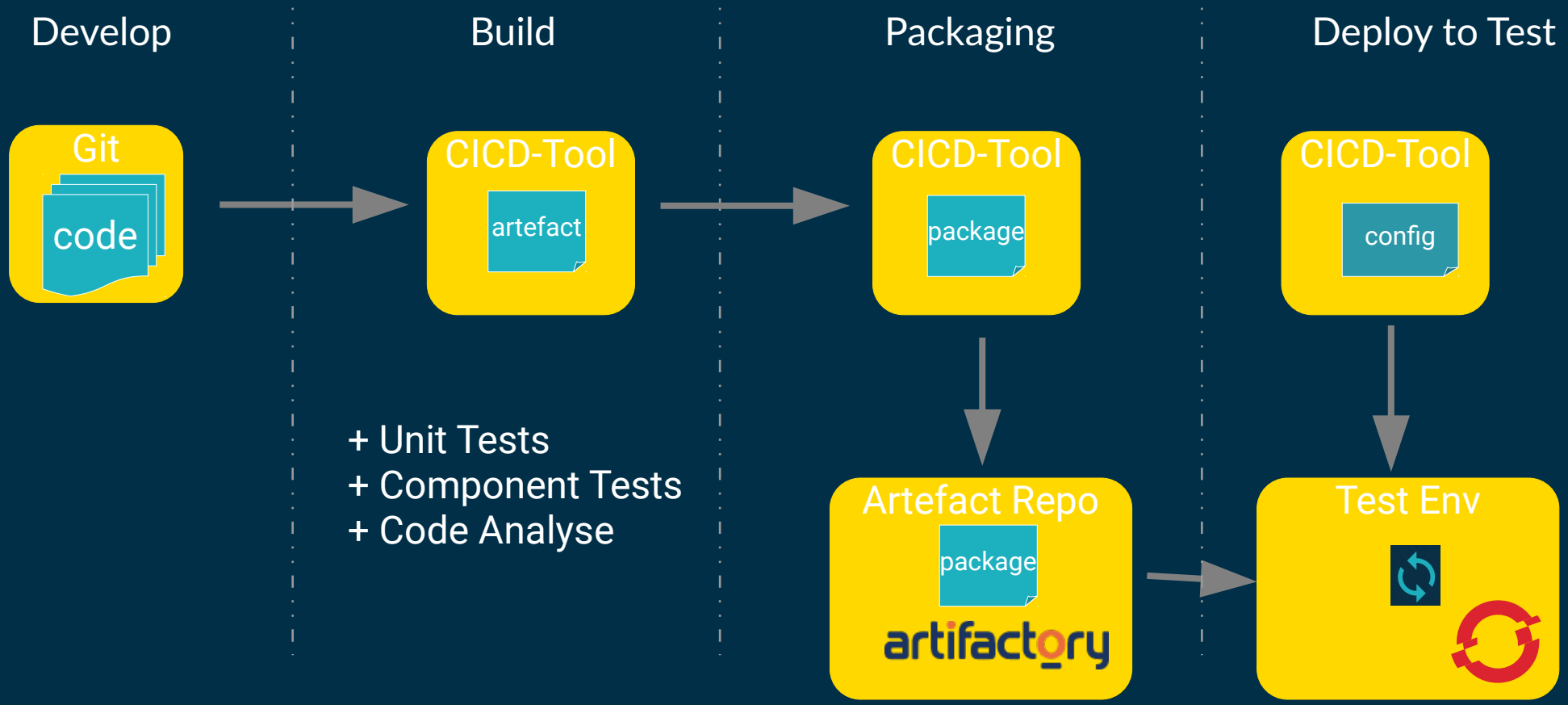


Continuous Delivery Pipeline

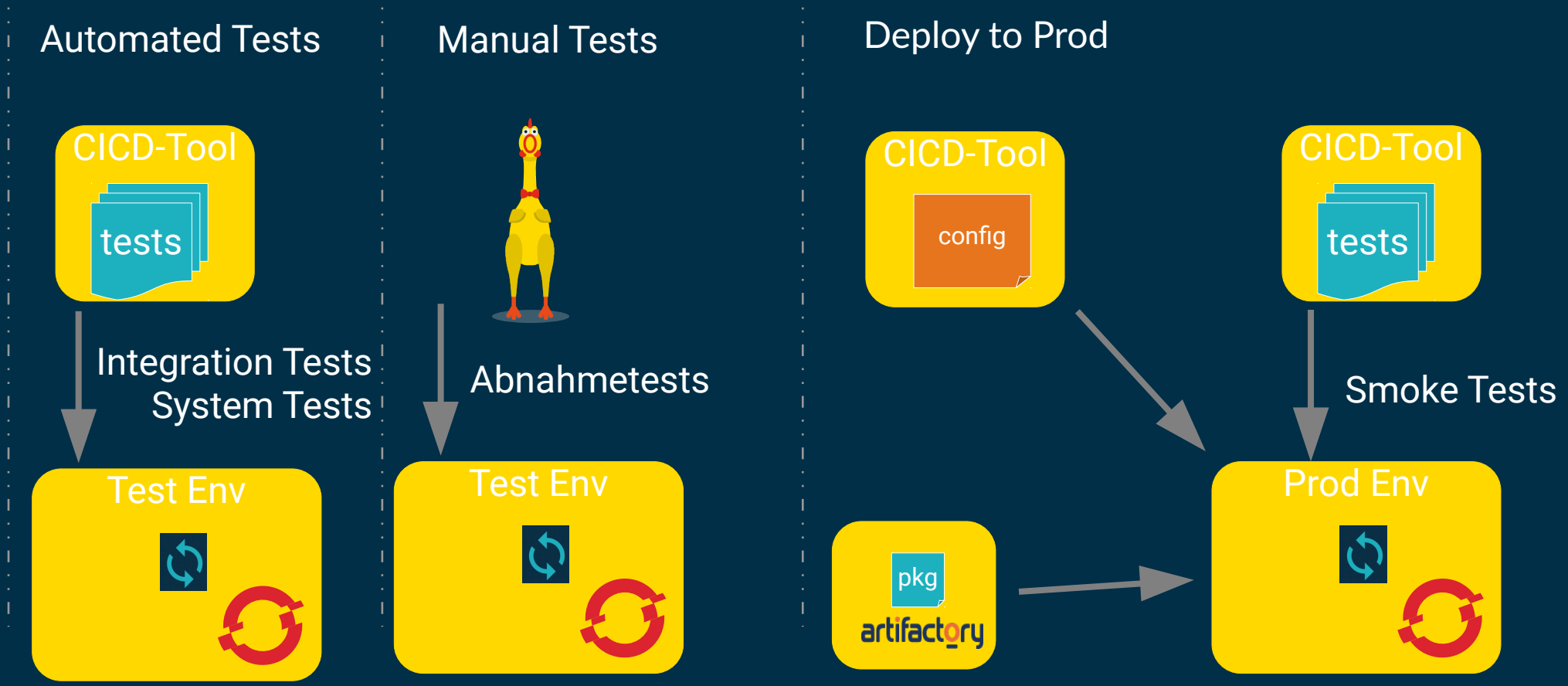
Highlevel CICD Pipeline

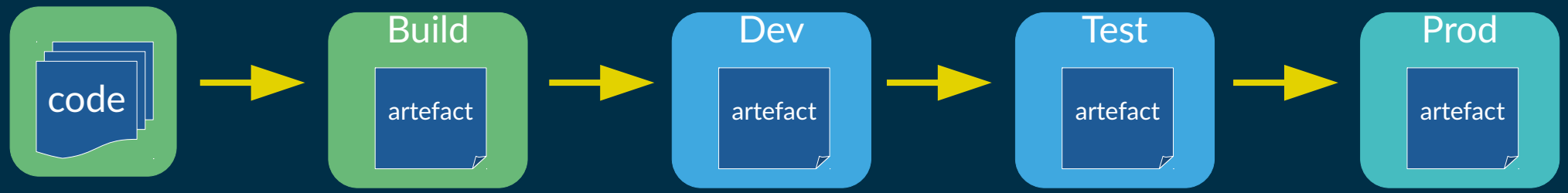


Example Pipeline (1)



Example Pipeline (2)





The **same** artifact is passed through the entire pipeline

Why should we use Pipelines

Pipeline Advantages

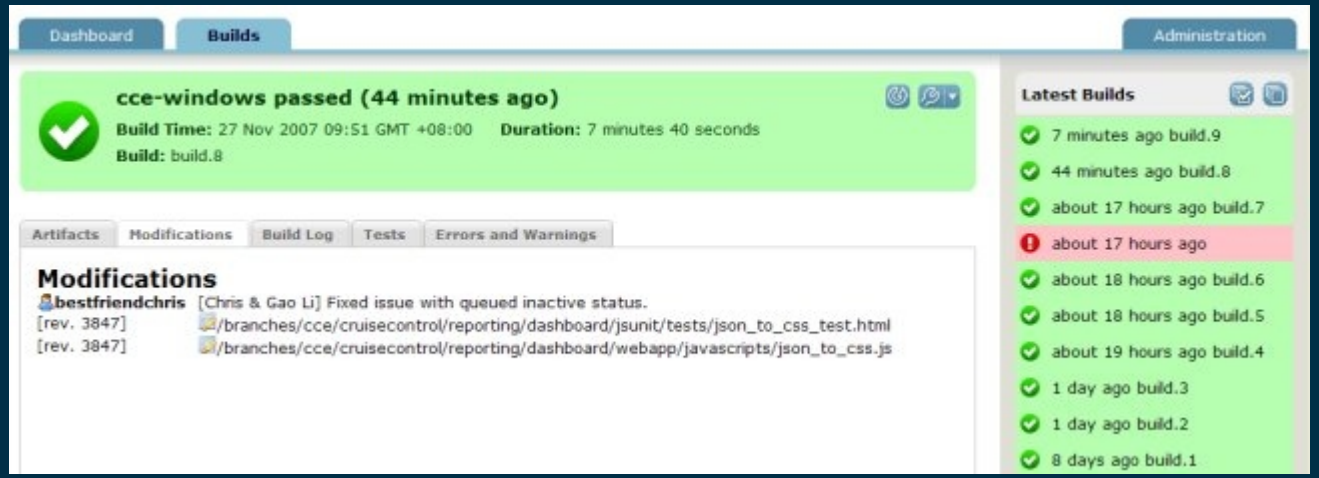
1. Can be reviewed, forked, iterated upon and audited
2. Running pipelines survive master restart
3. Can stop and wait for human input
4. Support complex CI/CD requirements
5. DSL can be extended through shared libraries

Questions?

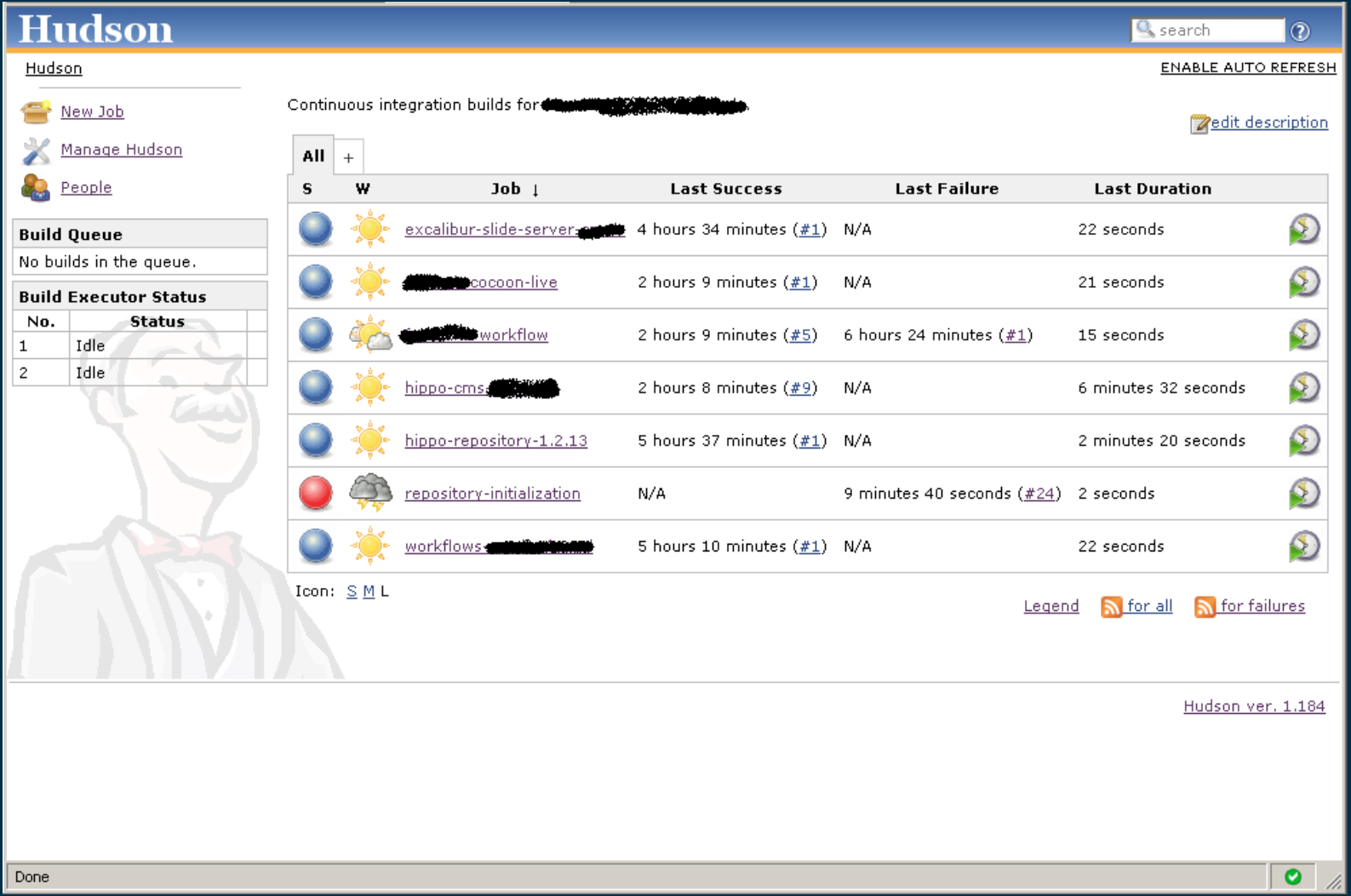
Jenkins

Once upon a time...

2001
CruiseControl



2005 Hudson


Hudson
search
Hudson
ENABLE AUTO REFRESH
New Job
Manage Hudson
People
Continuous integration builds for [redacted]
edit description
Build Queue
No builds in the queue.
Build Executor Status
No. Status
1 Idle
2 Idle
All +
S W Job ↓ Last Success Last Failure Last Duration
excalibur-slide-server-[redacted] 4 hours 34 minutes (#1) N/A 22 seconds
[redacted]cocoon-live 2 hours 9 minutes (#1) N/A 21 seconds
[redacted]workflow 2 hours 9 minutes (#5) 6 hours 24 minutes (#1) 15 seconds
hippo-cms-[redacted] 2 hours 8 minutes (#9) N/A 6 minutes 32 seconds
hippo-repository-1.2.13 5 hours 37 minutes (#1) N/A 2 minutes 20 seconds
repository-initialization N/A 9 minutes 40 seconds (#24) 2 seconds
workflows-[redacted] 5 hours 10 minutes (#1) N/A 22 seconds
Icon: S M L
Legend for all for failures
Hudson ver. 1.184
Done

2011 Jenkins

Jenkins | search | Claudia Meindl | log out

Jenkins › Rollout › Rollout-system › #36

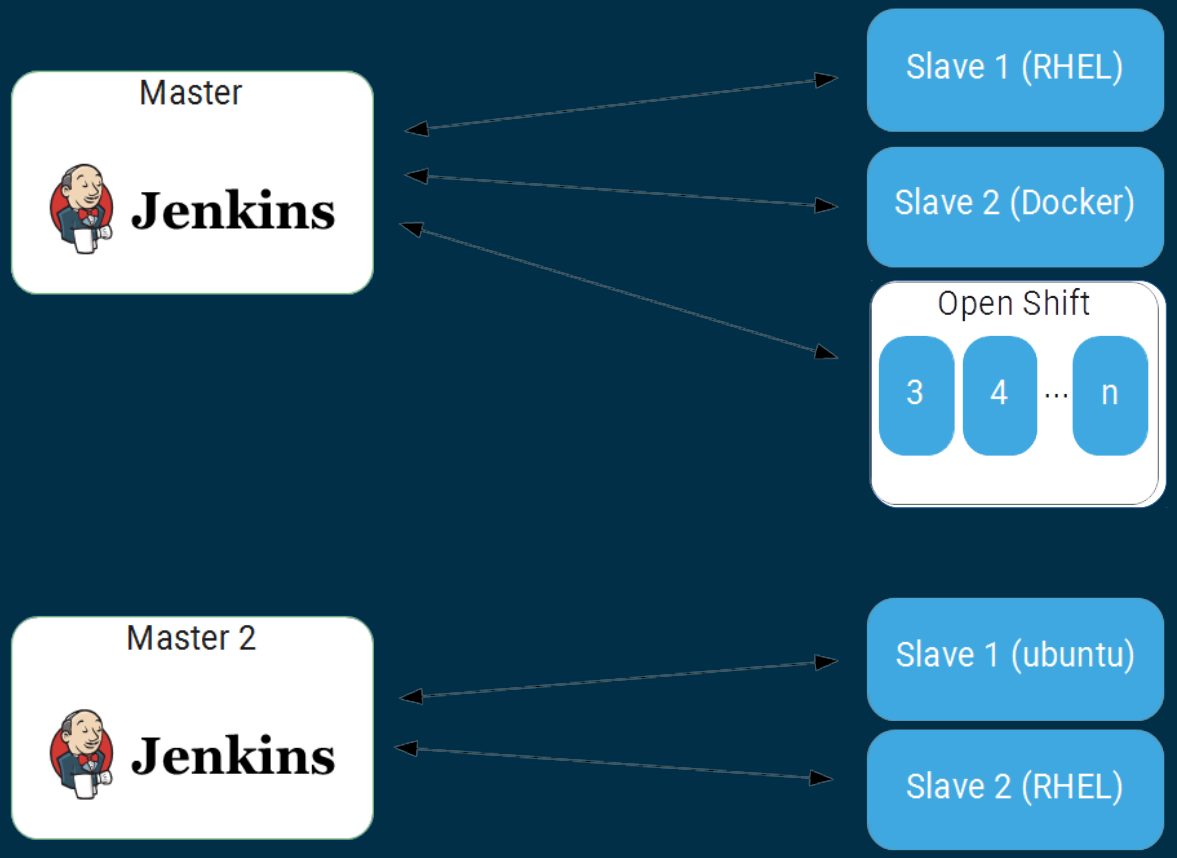
- Back to Project
- Status
- Changes
- **Console Output**
- View as plain text
- View Build Information
- Git Build Data
- Previous Build

Console Output

Skipping 1.196 KB.. Full Log

```
KoEttlJQNjBwdnF2sVayAwD4vuoDpQAAAA==[0mskipping: [teamwiesn.com]

TASK [smtp-server : include] ***************************************************
skipping: [teamwiesn.com]

TASK [ssl : include] ************************************************************
included: /var/lib/jenkins/jobs/Rollout-system/workspace/roles/ssl/tasks/setup.yml for
teamwiesn.com

TASK [ssl : install ssl packages] ***********************************************
ok: [teamwiesn.com] => (item=[u'ssl-cert', u'sslscan'])

TASK [ssl : Install ssl certificates] *******************************************
ok: [teamwiesn.com] => (item=teamwiesn.crt)

TASK [ssl : Install ssl keys] ***************************************************
ok: [teamwiesn.com] => (item=teamwiesn.key)

TASK [ssl : Generate dhparam key] ***********************************************
ok: [teamwiesn.com]

TASK [ssl : Check ssl forward secrecy key permission] ***************************
ok: [teamwiesn.com]

TASK [ssl : include] ************************************************************
skipping: [teamwiesn.com]

TASK [pip : Check to see if pip is already installed.] **************************
ok: [teamwiesn.com]
```

Architecture



Architecture

- Consists of Masters
  - Stateful
  - Management of Jobs, Credentials, ...
  - Plugins (managed on master)
- and Slaves
  - Actually run the jobs
  - Stateless

Scaling a Build Infrastructure

- jobs = number of developers * 3.333
- masters = number of jobs/500
- executors = number of jobs * 0.03

Source : User Handbook - Architecting for Scale

Jenkinsfiles

- Define your Pipeline
- Written in Groovy
  - Java based Scripring Language
- Stored on Jenkins-Master or in Git
  - Git is strongly prefered !

Jenkinsfiles : Declarative vs Scripted

Declarative

- Only DSL/Steps allowed
- script {} block for scripting

```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                echo 'Building...'
            }
        }
    }
}
```

Scripted

- Imperative Groovy script
- Deprecated

```
node {
    stage('Build') {
      echo 'Building...'
    }
}
```

Jenkinsfiles : Declarative Structure

1. pipeline keyword
2. Declarative Directives
   - Run on the Master
   - Set Pipeline Properties
3. Stages
   - Run on the slaves
4. Post Directives
   - Run on the Master or Slave
   - Conditional on Pipeline Status

```
pipeline {
    agent any
    parameters {
        string defaultValue: 'exampleParam', name: 'myParam'
    }
    environment {
        MY_ENV = "exampleEnv"
    }
    tools {
        maven 'Maven 3.5.0'
    }
    options {
        buildDiscarder logRotator(numToKeepStr: '5')
    }
    stages {
        stage('Hello') {
            steps {
                echo 'Hello'
            }
        }
        stage('World') {
            steps {
                sh 'echo World'
            }
        }
    }
    post {
        unsuccessful {
            echo 'Pipeline unsuccesful'
        }
    }
}
```

Execution

1. Setup
   1 Fetch and Validate Jenkinsfile
   2 Clones Shared Library (if defined)
   3 Parse Declarative Directives
   4 Clones Source Git Repository
2. Stages
   - Runs Pipeline Steps on Slaves
3. Post Directives
   - Cleanup
   - Notifications

Limitations

- Directive update on master
  - Triggers, Parameters, etc.
  - Pipeline needs to run once for master to get update
- Pipelines must survive restarts
  - Scripted Groovy Code gets transformed
  - Datastructures must be serelizable

Tools

Tools can be installed in two ways :

- Tool installer
  - Tool is installed on the slave at the start of the execution
  - Only supports selected tools (solution: custom-tools-plugin)
- Docker agent
  - Use container as execution environment

→ more in the labs

```
pipeline {
    agent any
    ...
    tools {
        maven 'Maven 3.5.0'
    }
    ...
}
```

```
pipeline {
    agent {
        docker { image 'node:14-alpine' }
    }
    ...
}
```

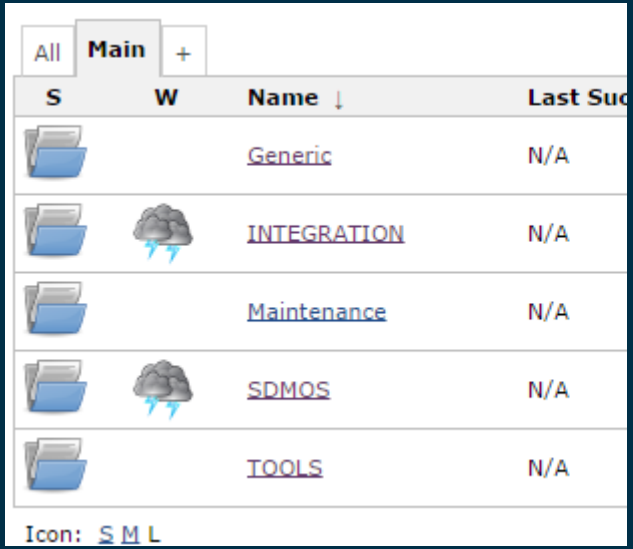
Best Practices

- Keep everything in version controll
- Organise Projects/Teams with folders
- Keep slaves stateless
- Reuse functionalities in shared libraries
- Debug with Replays
- Only one Source Repository Checkout

Best Practices : Folders

Organise Projects/Teams with folders

- Keep things organized
- Control Permissions
- Control Credentials Access



Best Practices : Stateless Slaves

Slaves should and Pipeline executions should be stateless

- Don’t depend on artifacts from previous execution
- Archive artifacts
- Clean up slave workspace
- Clean up oder builds

```
pipeline {
    …
    options {
        buildDiscarder logRotator(numToKeepStr: '5')
    }
    …
    stages {
        stage('Hello') {
            steps {
                sh 'mvn clean install'
            archiveArtifacts 'log/test.log'
            }
        }
    }
    post {
        unsuccessful {
            cleanWs()
        }
    }
}
```

Best Practices : Shared Libraries

Reuse functionalities in shared libraries

- Use same workflow for similar projects
  - eg. all spring boot applications
- Put common functionallity in shared library
  - Easyer to maintain and test and update

Example : [Puzzle Jenkins Shared Library (GPL-3.0)]

Best Practices : Debug with Replays

Use Replays to :

- View Jenkinsfiles of past executions
- Try out changes without commiting to git

→ Remember to persist your canges in version control !

Best Practices : Only one Checkout

Disable default declarative checkout if you explicitly checkout later.

```
pipeline {
    …
    options {
        skipDefaultCheckout true
    }
    …
    stages {
        stage('Setup') {
            steps {
                git(credentialsId: 'mysecret'
                    url: 'https://github.com/myorg/myrepo.git')
            }
        }
    }
    ...
}
```

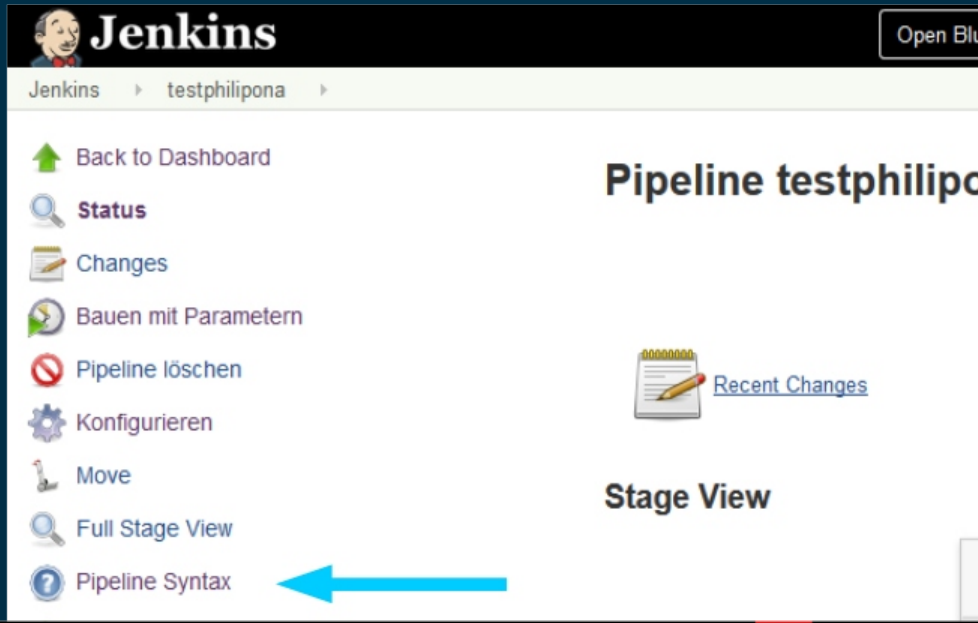
Resources

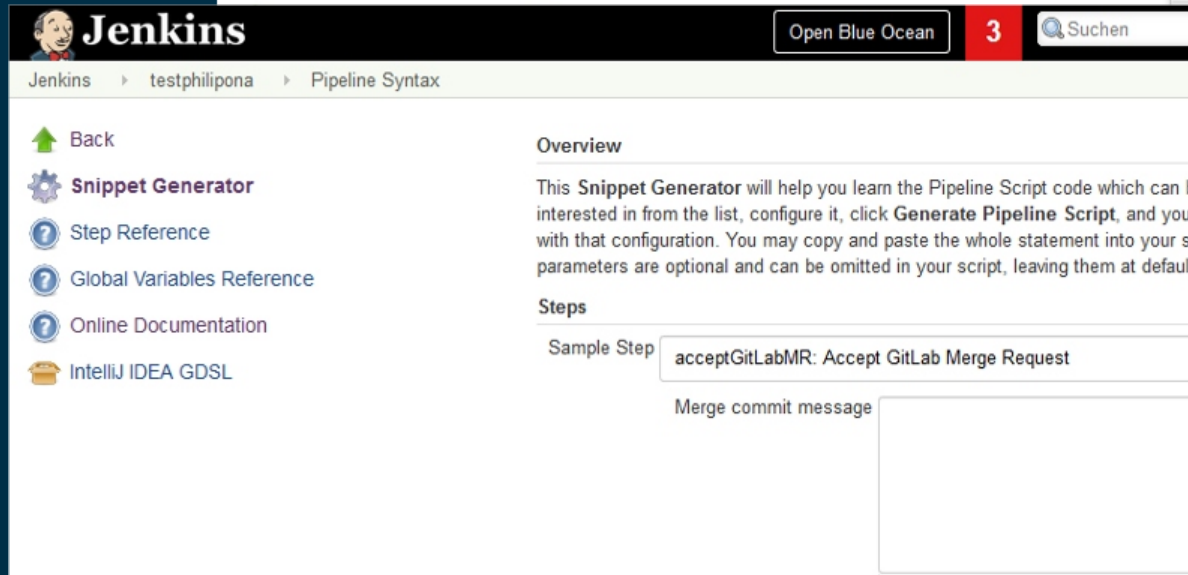
In Jenkins :

- Snipet/Directive generator
- Jenkins Step reference
- Shared Library

Online :

- Jenkins User Handbook
  - Step reference
- Groovy
  - Collections





Questions?