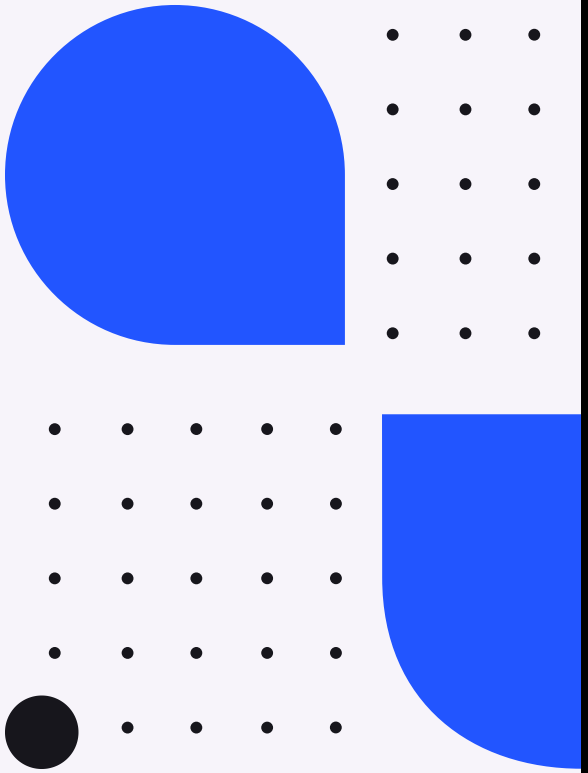


Container Extensions für z/OS (zCX)

Maike Havemann
IBM Z Technical Specialist

20.10.2021 / zExpertenforum

IBM Z

z/OS Container Extensions

.. was ist zCX?

Use Cases

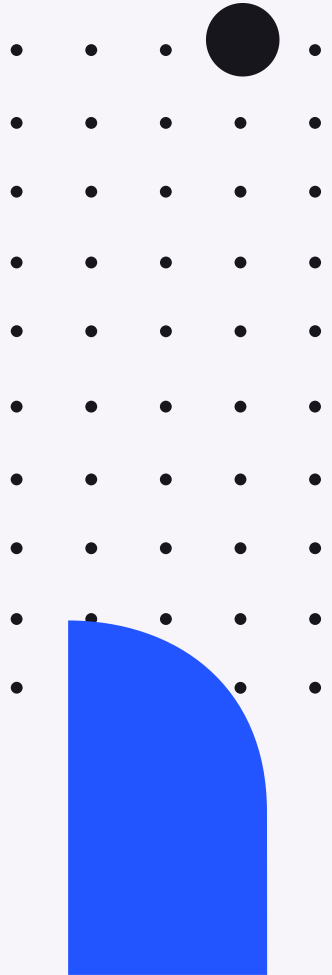
.. was theoretisch geht

DevOps

.. wenn es innotivativ sein soll

Praxis

.. was in der Realität umgesetzt wird



z/OS Container Extensions

ab V2.4

- **Z Linux Software als Docker Container in z/OS**

  ohne separaten Linux Server

- **Maintenance über z/OS und z/OS Qualities of Service**

- **z14 oder z15 mit Container Hosting Foundation**

  HW FC 0104 oder IBM Container Hosting Foundation für z/OS

- **zCX 90-Tage-Testversion**

  APAR (OA58969)

z/OS Container Extensions

ab V2.4

Linux Docker Appliance

Maintenance und IBM Support, Provisionierung durch z/OSMF Workflows

Standard Docker Interfaces

Jede Software, die als Docker Image auf Linux auf Z verfügbar ist // IBM Z Registry
Kommunikation mit native z/OS Applikationen über high speed virtual IP network
Keine z/OS Skills für Entwicklung und Deployment von Docker Containern

Keine Linux Admin Skills notwendig

Interfaces auf Docker CLI limitiert, kein Zugriff auf Linux Kernel

Gemanaged als z/OS Prozess

Mehrere Instanzen in einem z/OS System
zCX Workloads sind zIIP eligible

Use Cases

Erweiterung z/OS Applikationen

- Neuste Microservices (logstash, Etcd, Wordpress, etc.)
- No-SQL DBs (MongoDB, IBM Cloudant, etc.)
- Analytics Frameworks
- Messaging Frameworks (z.B. Apache Kafka)
- Web Server Proxies (z.B. nginx)
- Neuste Programmiersprachen und -umgebungen

System Management

- System Management Komponenten (bisher nicht verfügbar für z/OS)
- z.B.
  - Tivoli Enterprise Portal (TEPS)
  - Service Management Unite (SMU)

Open Source

- Ergänzen eines bestehenden z/OS, Zowe und DevOps
- Gitlab/Github server
- Linux basierte Dev Tools
- Linux Shell Environments
- ...

Use Cases - Redbook

Apache Kafka / ZooKeeper

Open Source Messaging System, Echtzeit-Daten

IBM App Connect Enterprise

Integrationsserver für z/OS Subsysteme mit IMS, DB2, CICS & MQ

IBM Aspera fasp.io Gateway

TCP/IP Tunnel für highspeed Datentransport, bspw. MQ

IBM MQ Client Concentrator

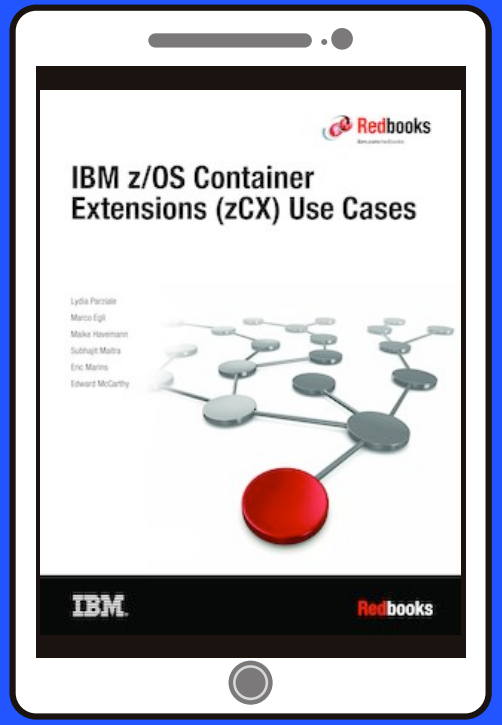
Direkte Kommunikation mit MQ Queque Managers im z/OS

DevOps

Simpler DevOps Flow

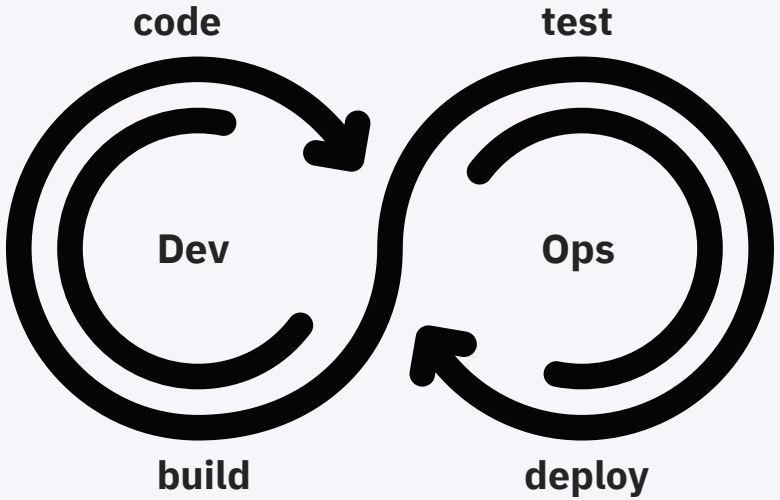
Monitoring mit SMU

UI + Dashboards, Monitoring IBM Z

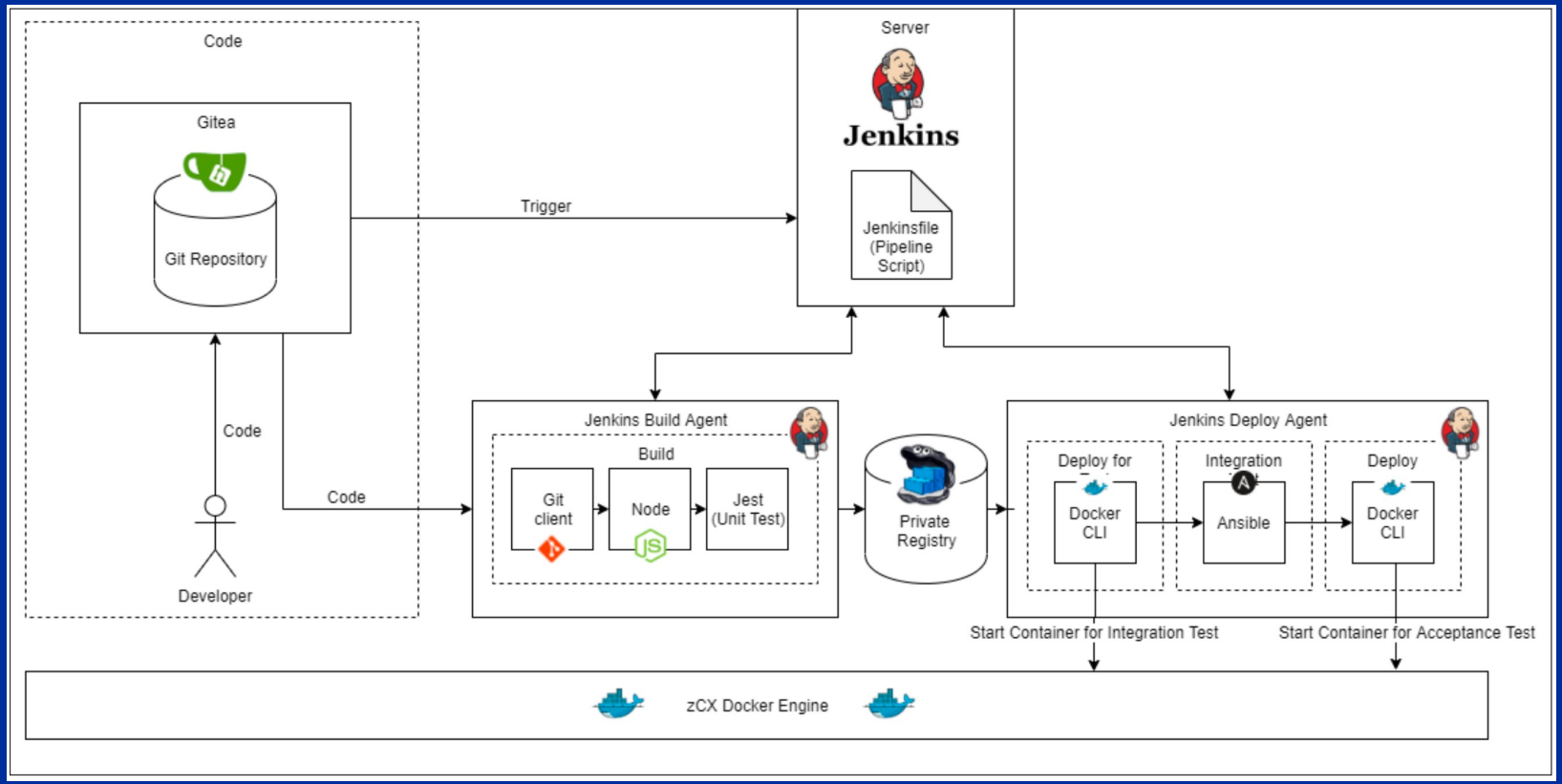


- Menschen > Prozesse, Tools
- CI/CD
- Fehlerreduzierung
- Plattform-unabhängig

DevOps



DevOps Use Case / Architektur Überblick



DevOps / Gitea als Code Repository

Was ist Gitea?

- Self-hosted Git Service
- Für viele Plattformen / Architekturen
- Schnell und intuitiv aufsetzbar

Anforderungen

- Laufende zCX Instanz
- Volume für Gitea angelegt
- Laufender Gitea Container
- Code für eine node.js Applikation

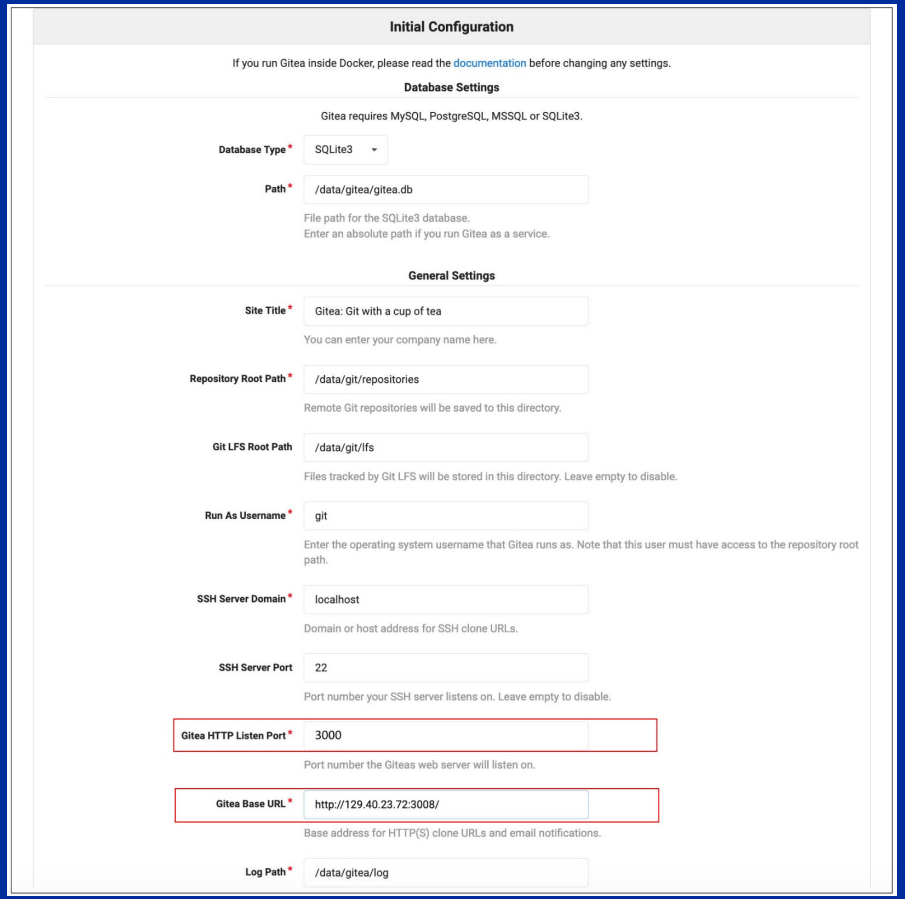
Instruktionen: siehe Redbook

DevOps Use Case / Gitea als Code Repository

1. Gitea User Interface über Browser öffnen
2. Anmelden

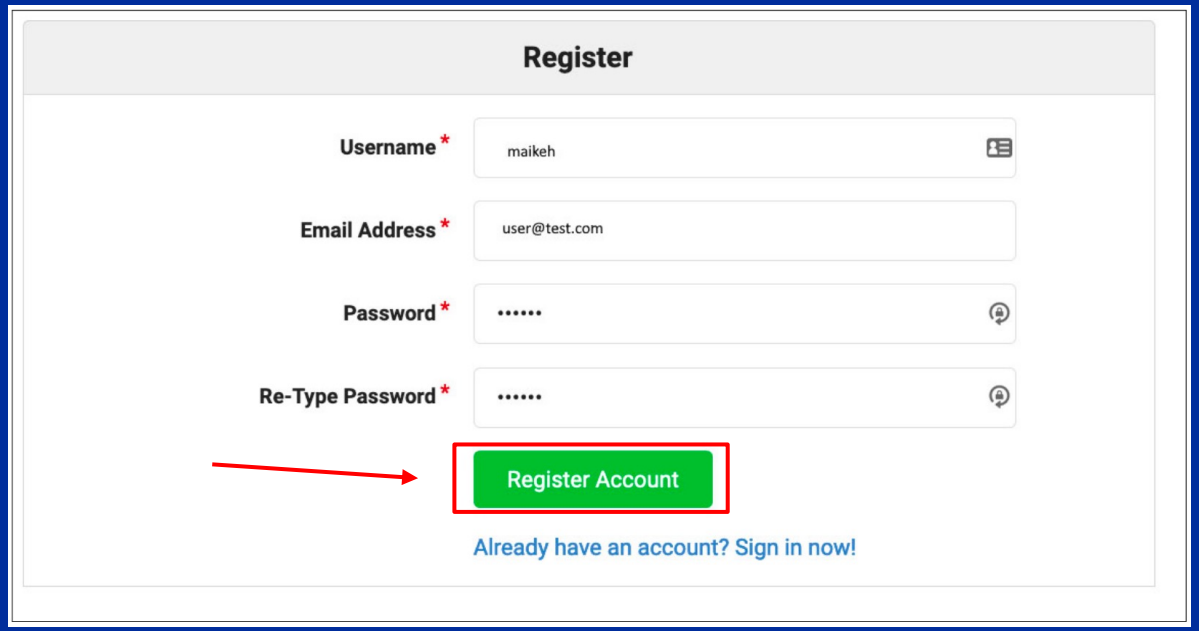
DevOps Use Case / Gitea als Code Repository

3. Gitea Container einmalig konfigurieren



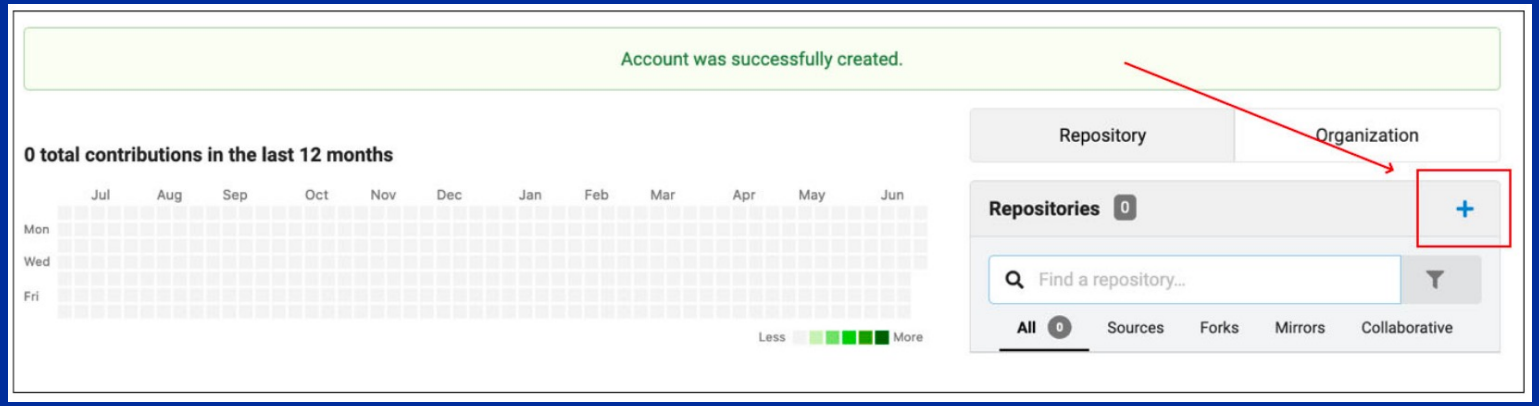
DevOps Use Case / Gitea als Code Repository

4. Accountinformationen ausfüllen und registrieren



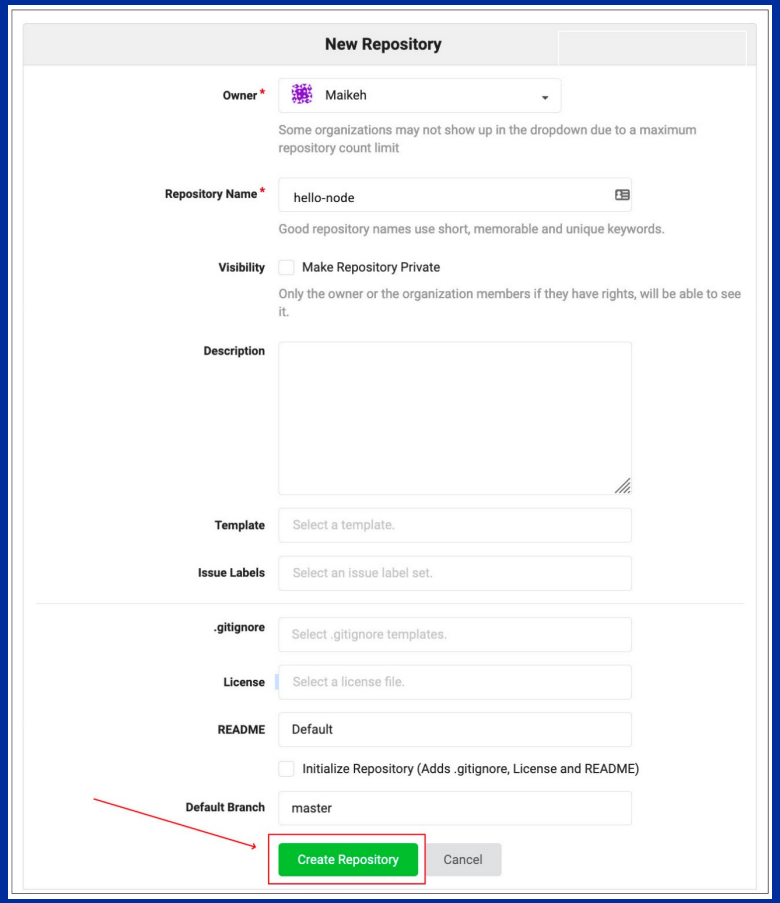
DevOps Use Case / Gitea als Code Repository

5. Repository für den Code einer Applikation anlegen



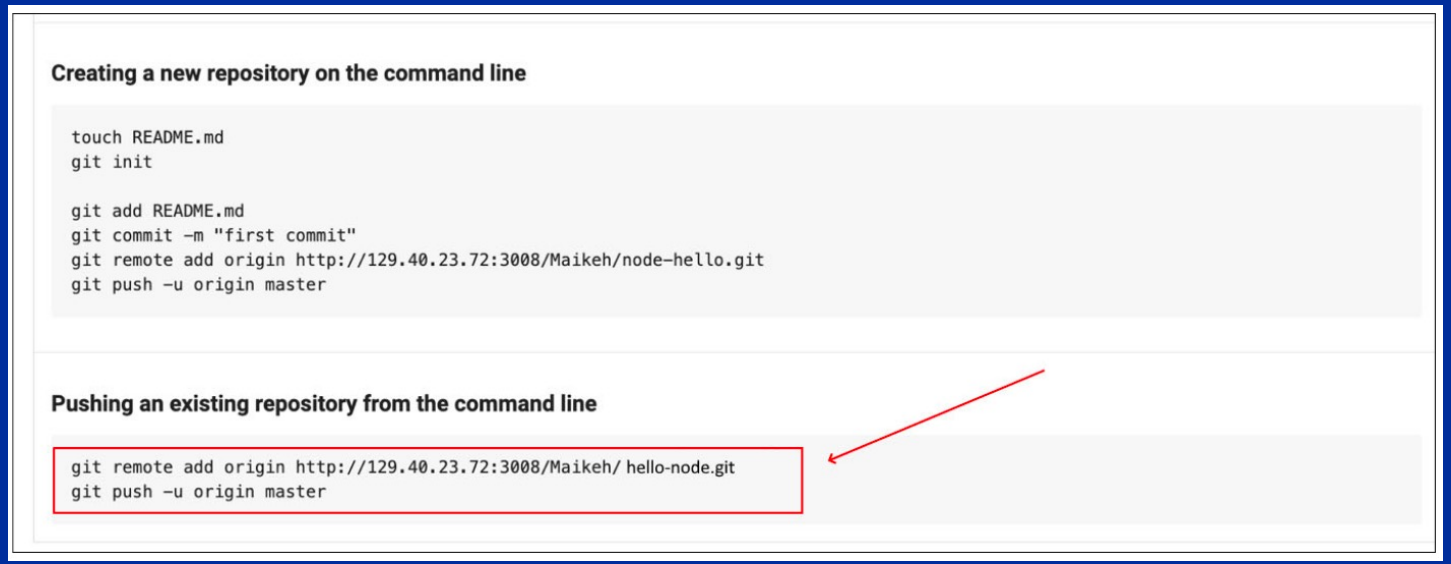
DevOps Use Case / Gitea als Code Repository

5. Repository für den Code einer Applikation anlegen



DevOps Use Case / Gitea als Code Repository

6. Instruktionen um Code ins Repository hinzuzufügen



Creating a new repository on the command line

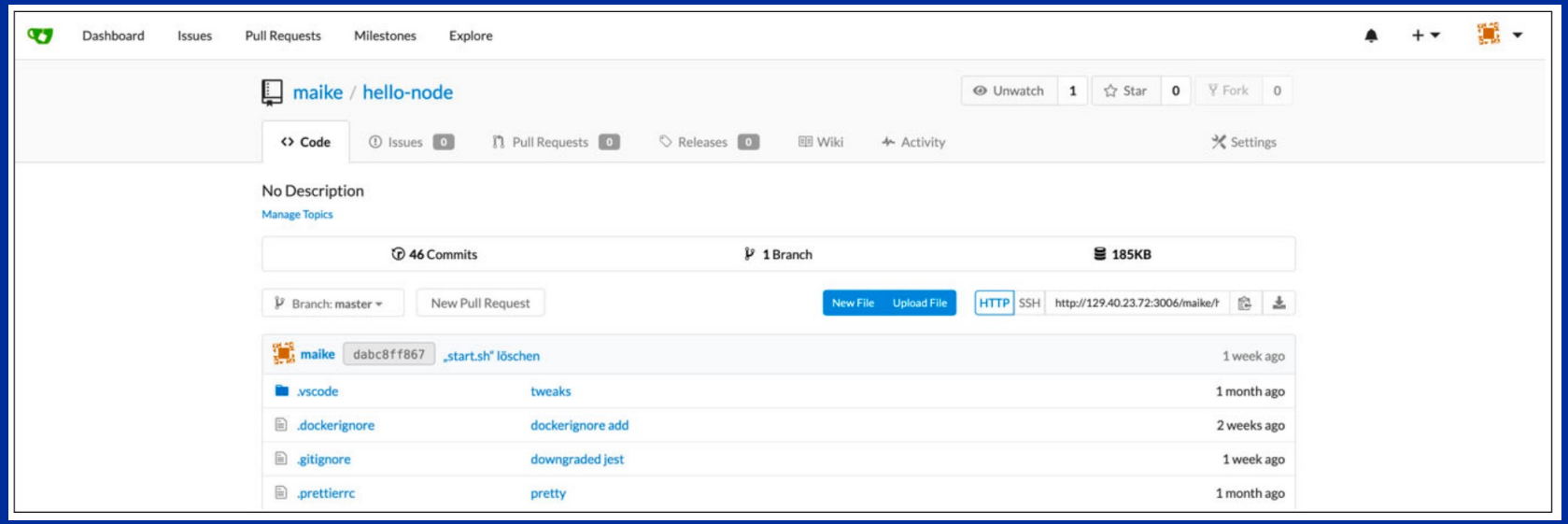
```
touch README.md
git init

git add README.md
git commit -m "first commit"
git remote add origin http://129.40.23.72:3008/Maikeh/node-hello.git
git push -u origin master
```

Pushing an existing repository from the command line

```
git remote add origin http://129.40.23.72:3008/Maikeh/ hello-node.git
git push -u origin master
```

DevOps Use Case / Gitea als Code Repository

7. Node.js Applikation aus dem Anhang vom Redbook in Repository pushen
Achtung: Ein Dockerfile wird benötigt



DevOps / Jenkins für automatisierte Builds

Was ist Jenkins?

- Open Source Server für eine robuste CI/CD Umgebung
- Ausführen von Jobs durch Jenkins Nodes
- Erstellen und Ausführen eines DevOps Flows durch Pipelinescript

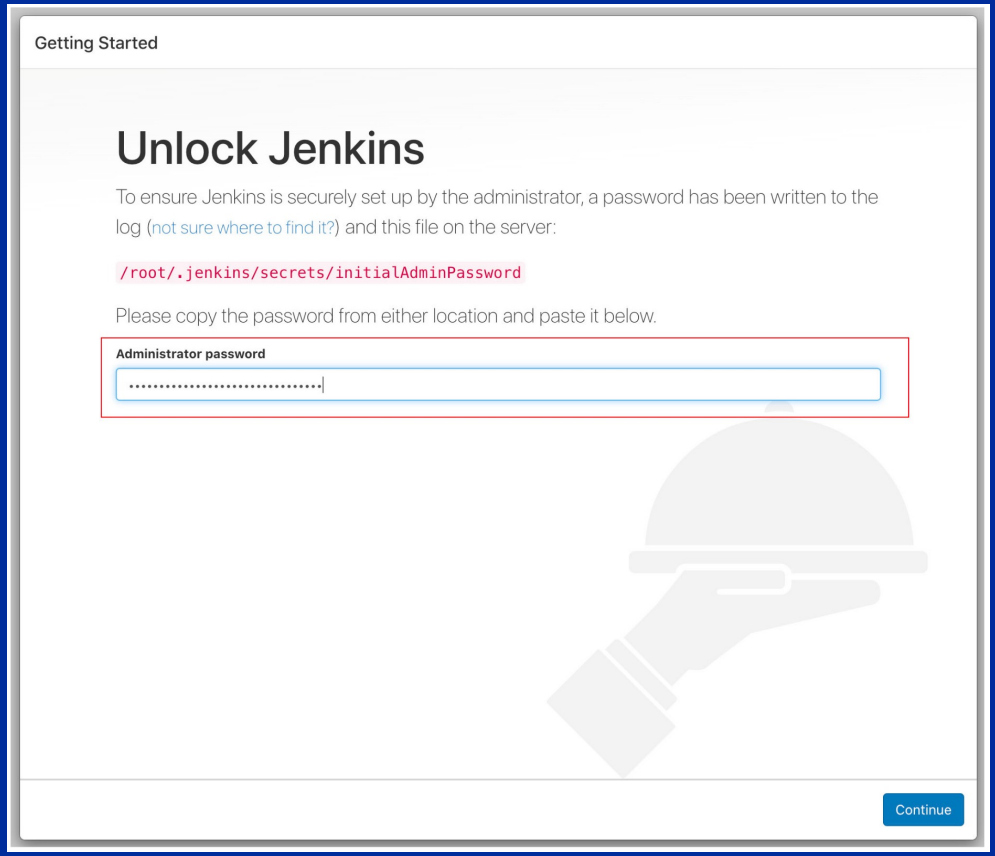
Anforderungen

- Laufende zCX Instanz
- Volume für Jenkins angelegt
- Laufender Jenkins Container
- Admin Passwort (docker logs)

Instruktionen: siehe Redbook

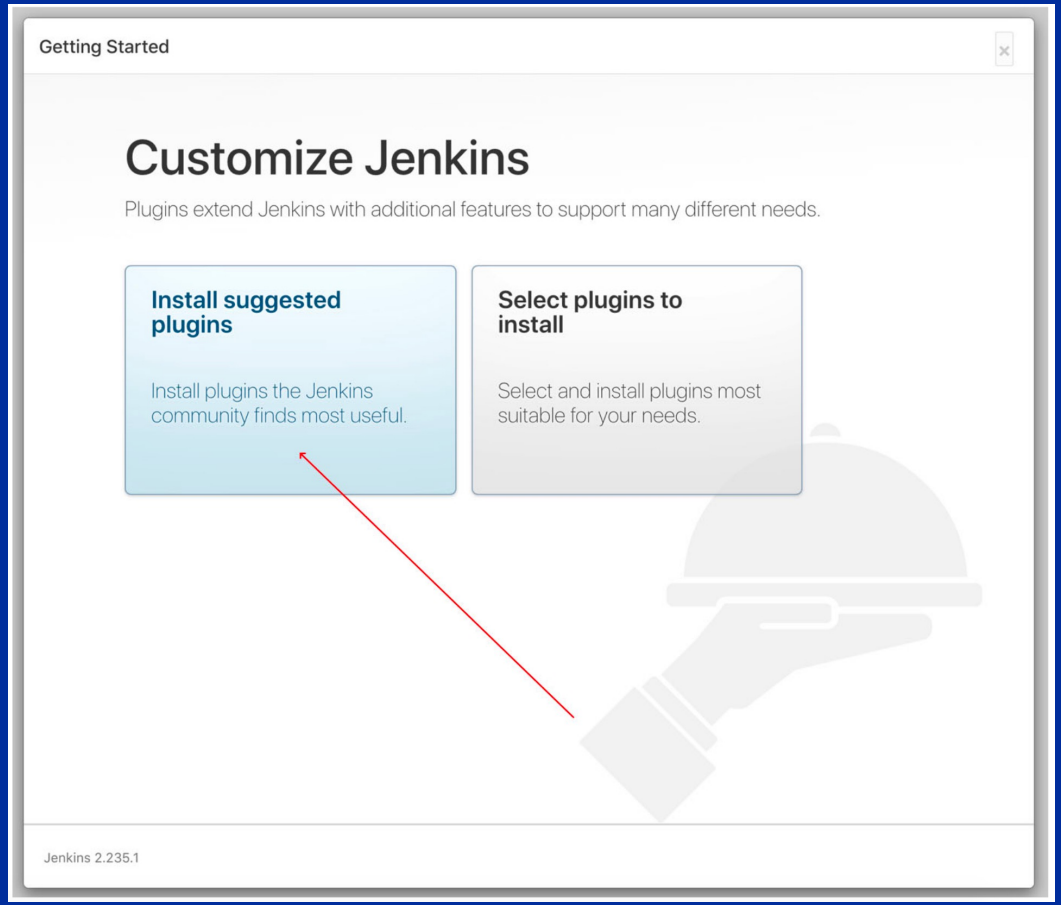
DevOps Use Case / Jenkins für automatisierte Builds

1. Jenkins User Interface über Browser öffnen
2. Admin Passwort aus Docker Konsole einfügen



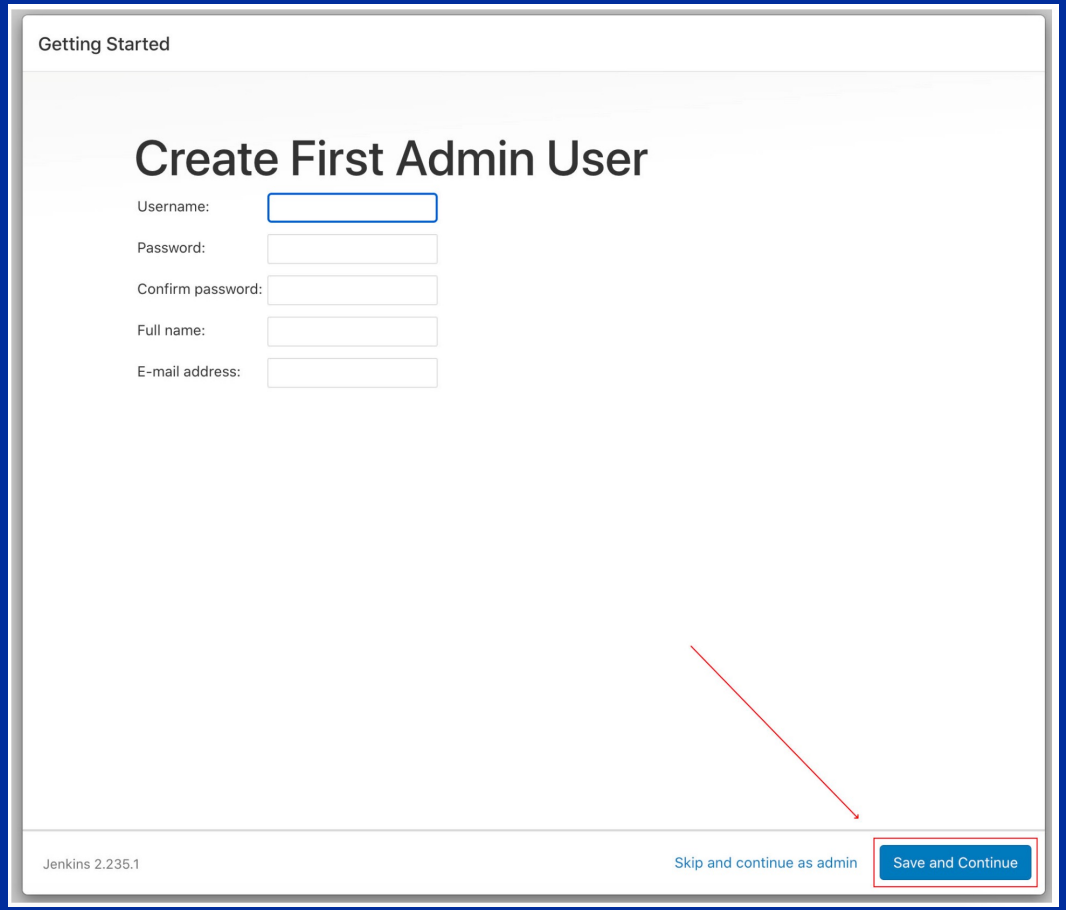
DevOps Use Case / Jenkins für automatisierte Builds

3. Vorgeschlagene Plugins installieren



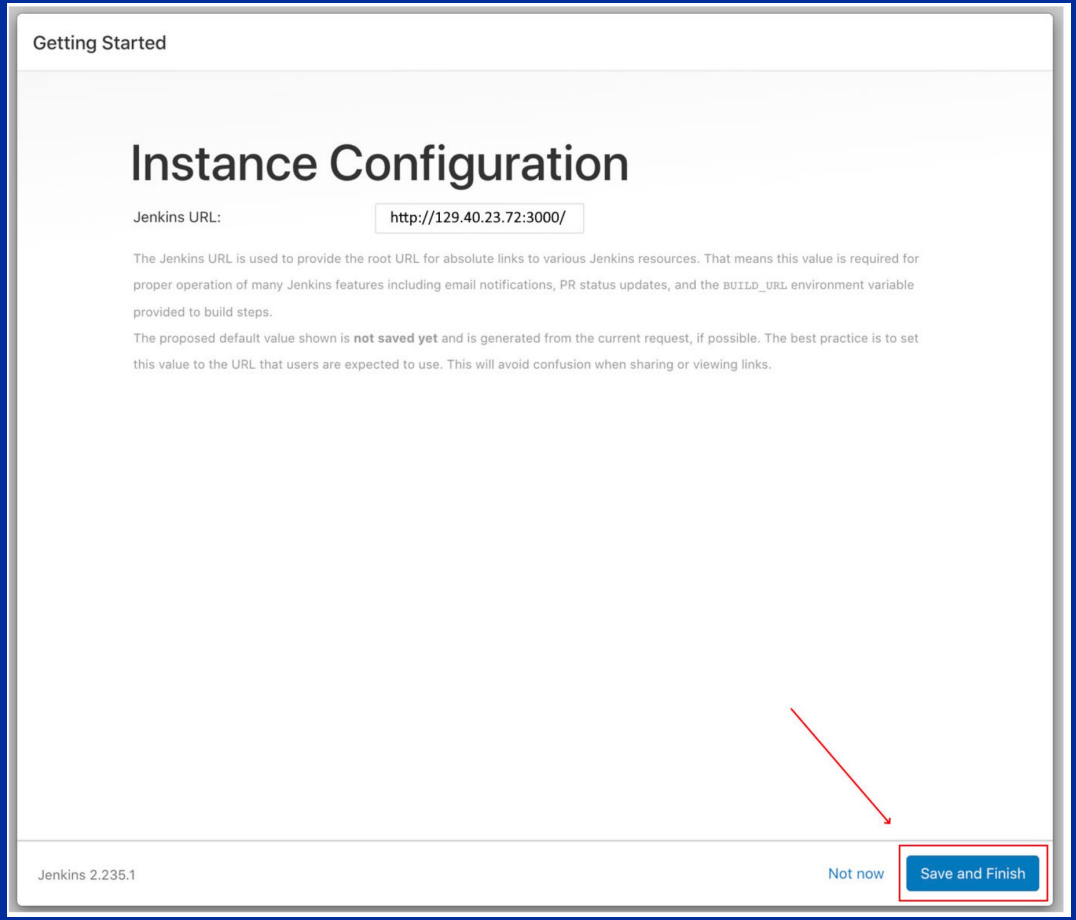
DevOps Use Case / Jenkins für automatisierte Builds

4. Admin User erstellen



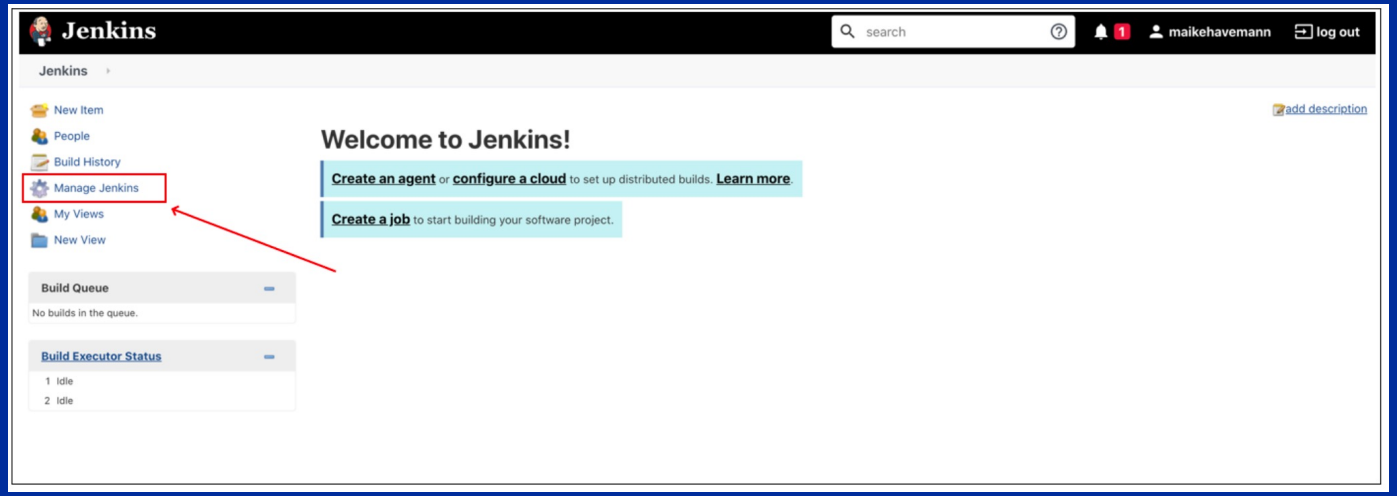
DevOps Use Case / Jenkins für automatisierte Builds

5. Jenkins URL in die Instanz Konfiguration einfügen



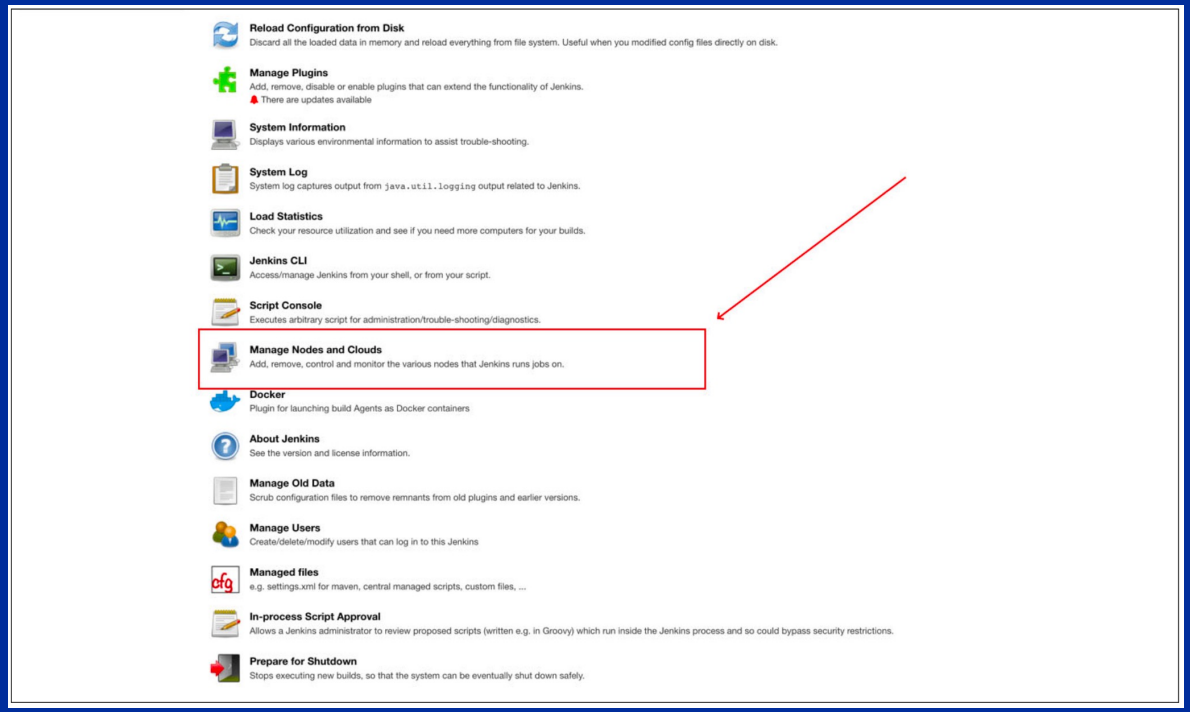
DevOps Use Case / Jenkins für automatisierte Builds

6. Ins Manage Jenkins Menü gehen



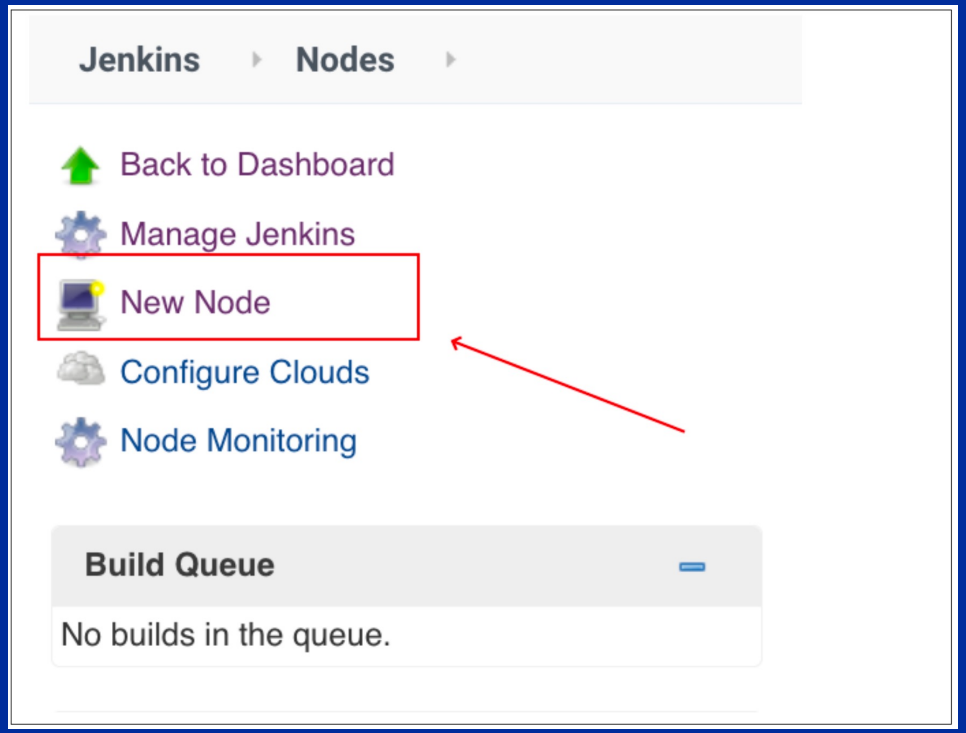
DevOps Use Case / Jenkins für automatisierte Builds

7. Auf *Manage Nodes and Clouds* klicken



DevOps Use Case / Jenkins für automatisierte Builds

8. Neuen Node anlegen



DevOps Use Case / Jenkins für automatisierte Builds

9. Build Node konfigurieren

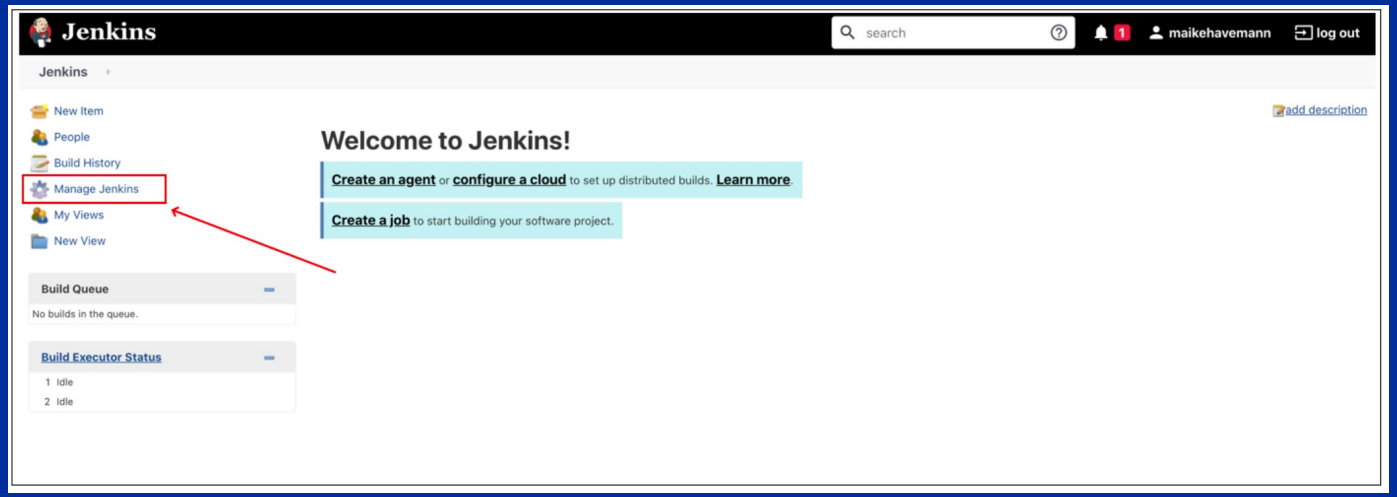
DevOps Use Case / Jenkins für automatisierte Builds

10. Node Secret kopieren um den Docker Container für den Jenkins Node zu starten

DevOps Use Case / Jenkins für automatisierte Builds

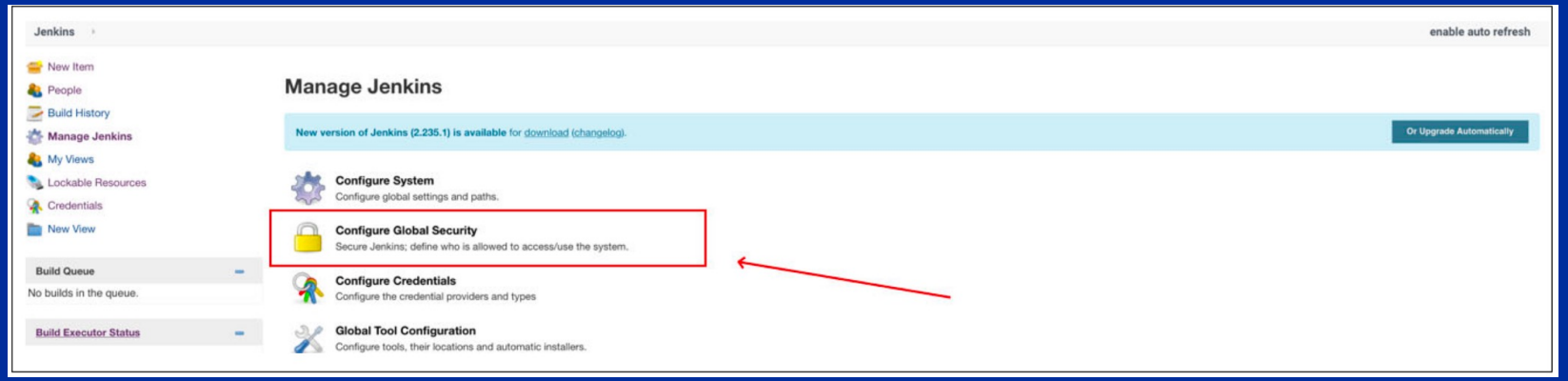
Kommunikation zwischen Jenkins Server und Build Node sicherstellen

11. Ins Manage Jenkins Menü gehen



DevOps Use Case / Jenkins für automatisierte Builds

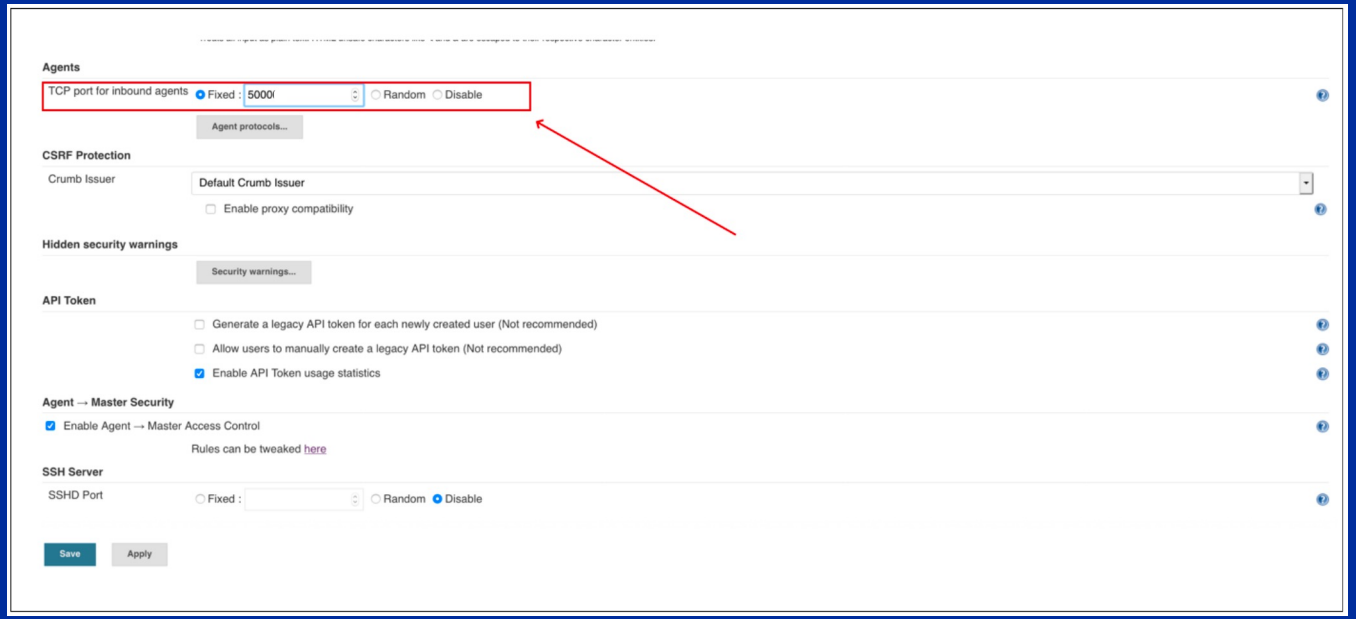
12. Sicherheitseinstellungen konfigurieren



DevOps Use Case / Jenkins für automatisierte Builds

13. TCP Port auf 5000 (private registry) setzen und den Port beim Start vom Node Container mitgeben
14. Jenkins Build Node Container starten

Instruktionen: siehe Redbook



DevOps / Ansible für automatisiertes Deployment und Tests

Was ist Ansible

- Automations Engine für z.B. Deployment von Applikationen
- Einfaches Deployment durch YAML Playbooks
- Kann standalone oder als Plugin in Jenkins verwendet werden

Anforderungen

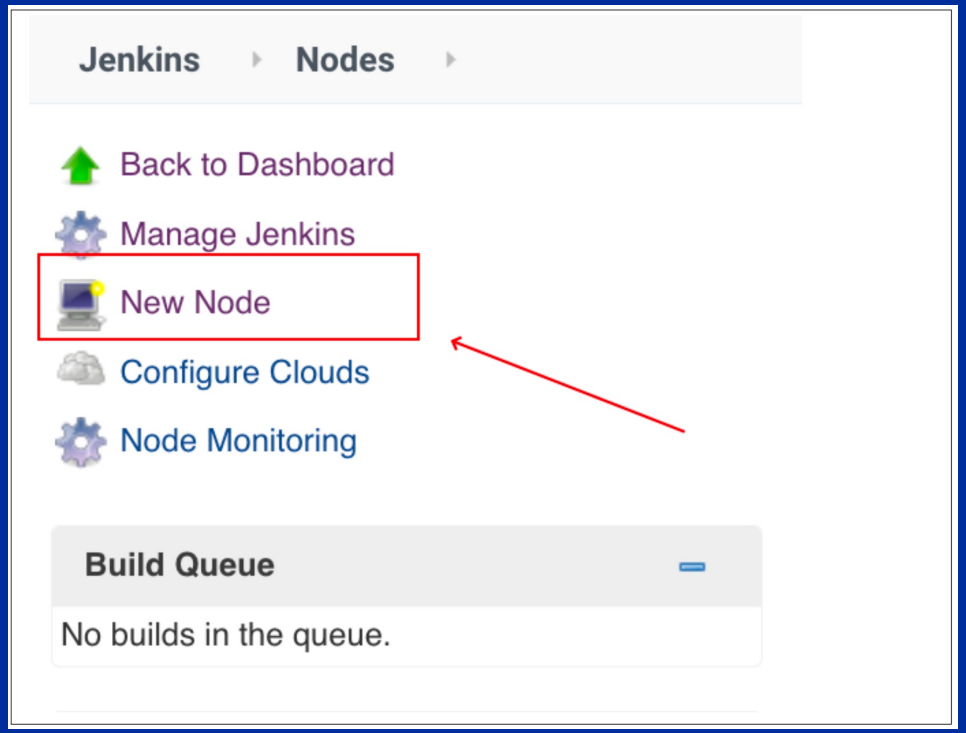
- Laufende zCX Instanz
- Laufender Jenkins Container
- Ansible Playbooks für Tests und Deployment (abgelegt im Gitea Repository)

Instruktionen: siehe Redbook

DevOps Use Case / Ansible für automatisiertes Deployment und Tests

Ansible in Jenkins als Deploy Node aufsetzen

1. Ins Manage Jenkins Menü gehen
2. Auf Manage Nodes and Clouds klicken
3. Neuen Node anlegen



DevOps Use Case / Ansible für automatisiertes Deployment und Tests

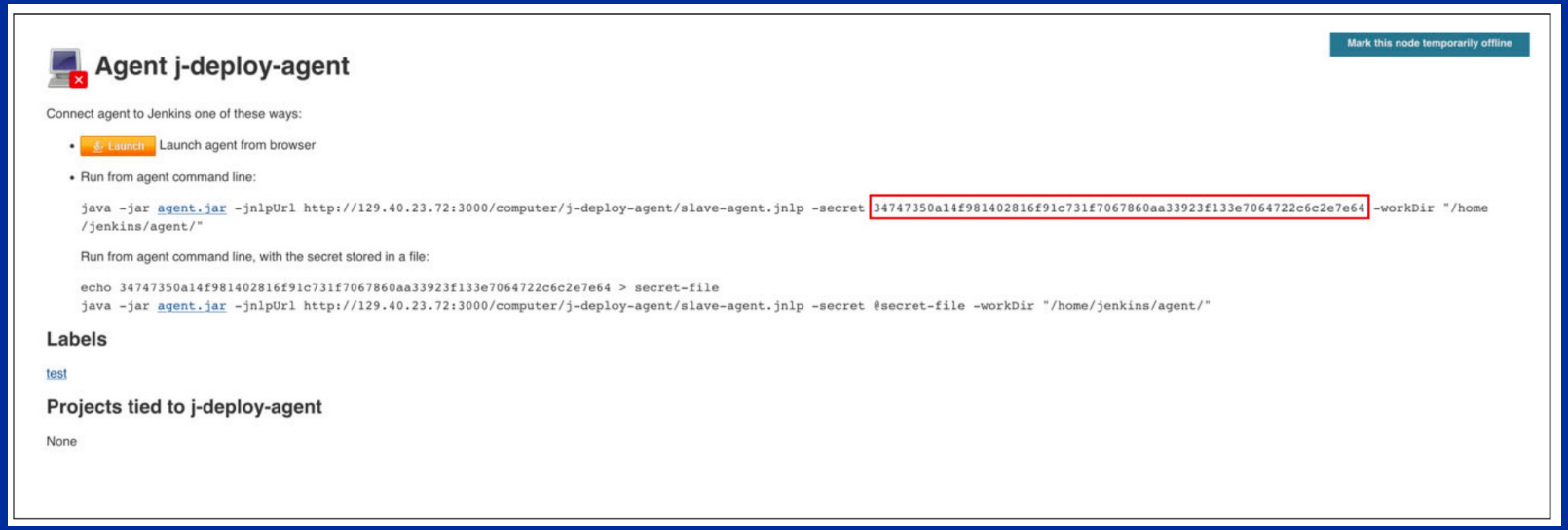
Ansible in Jenkins als Deploy Node aufsetzen

4. Jenkins Deploy Node konfigurieren

DevOps Use Case / Ansible für automatisiertes Deployment und Tests

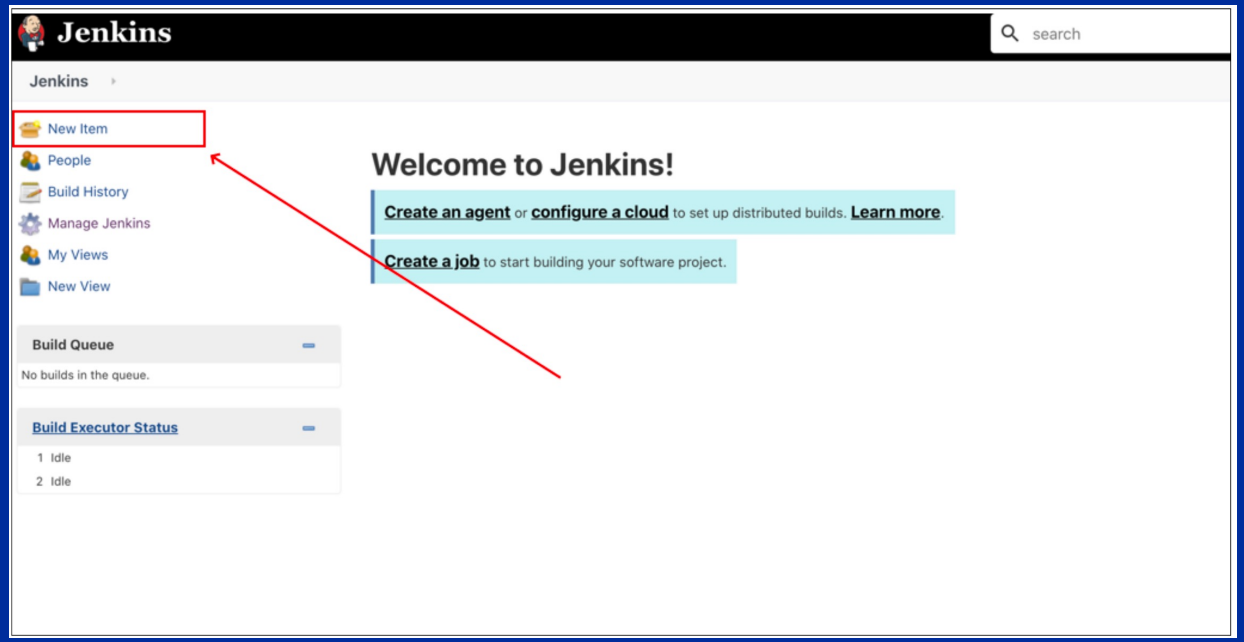
Ansible in Jenkins als Deploy Node aufsetzen

5. Node Secret kopieren und den Docker Container für den Deploy Node zu starten
Instruktionen: siehe Redbook



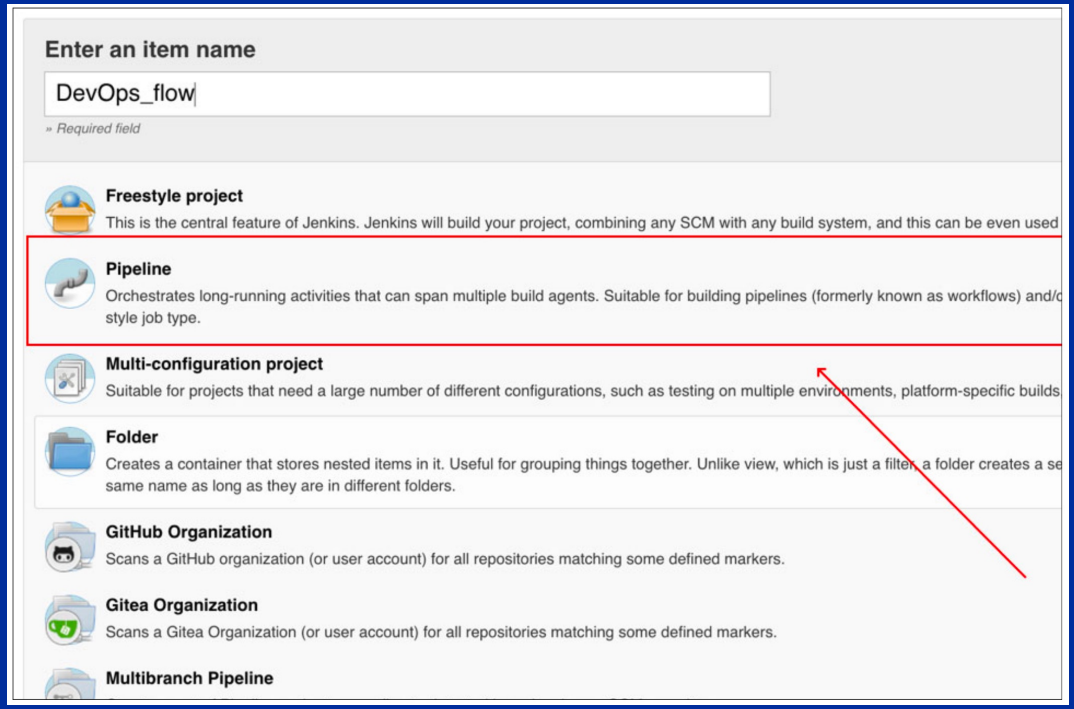
DevOps Use Case / Erstellen einer Pipeline

1. Neues Item im Jenkins Interface anlegen



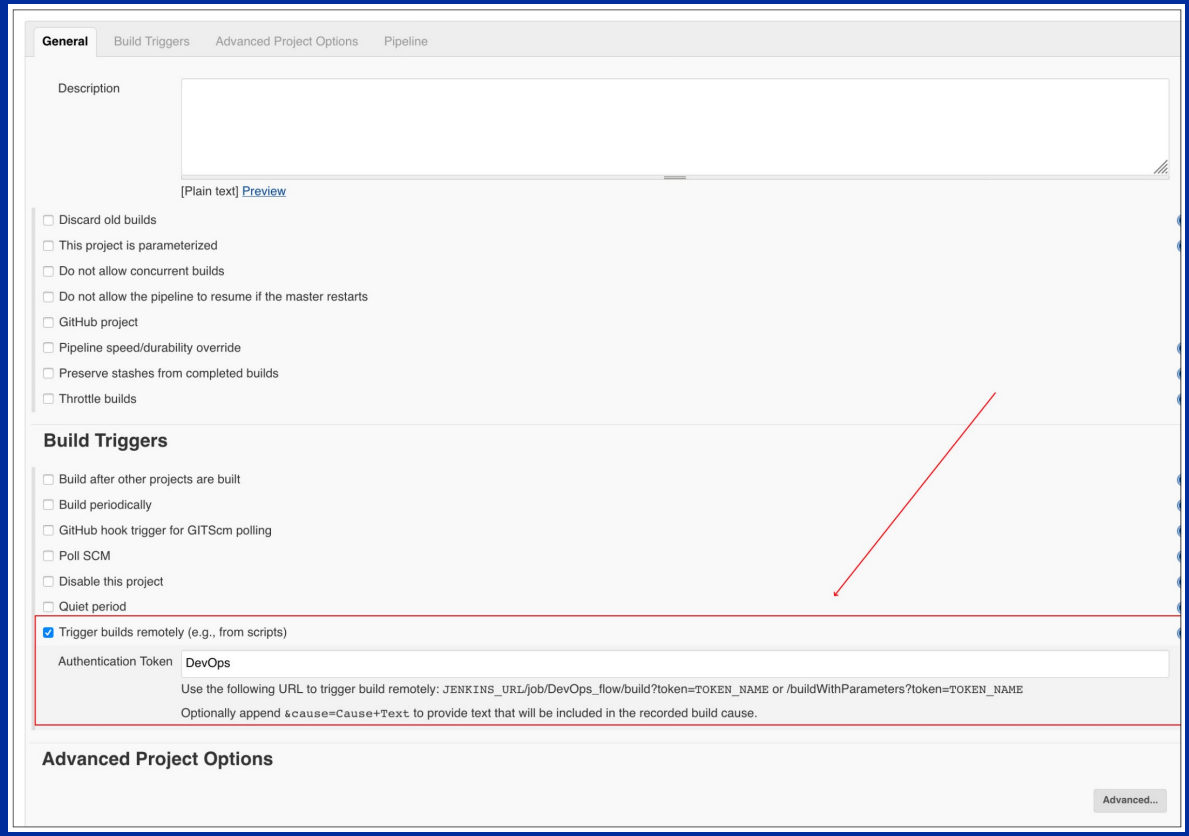
DevOps Use Case / Erstellen einer Pipeline

2. Item benennen und Pipeline auswählen



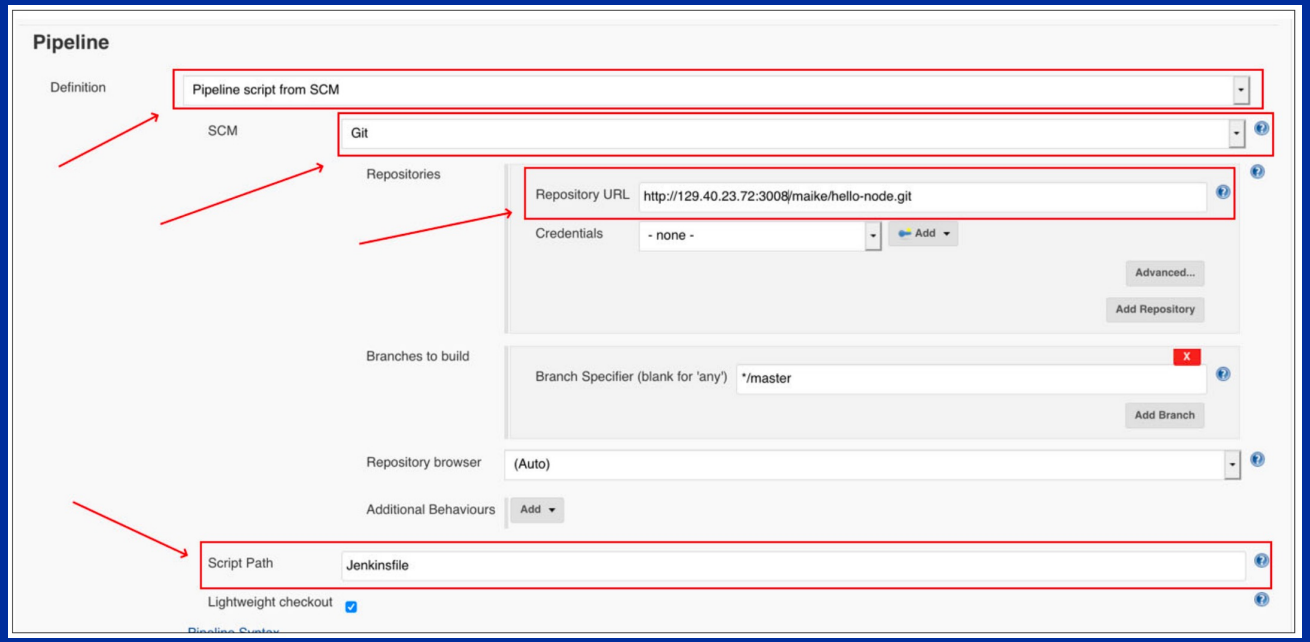
DevOps Use Case / Erstellen einer Pipeline

3. Aktivieren eines Triggers um mit jedem Code push auf die Applikation die Pipeline auszulösen



DevOps Use Case / Erstellen einer Pipeline

4. Steuerung der Pipeline durch Skript (“Jenkinsfile”), abgelegt in Gitea Repository



DevOps Use Case / Erstellen einer Pipeline

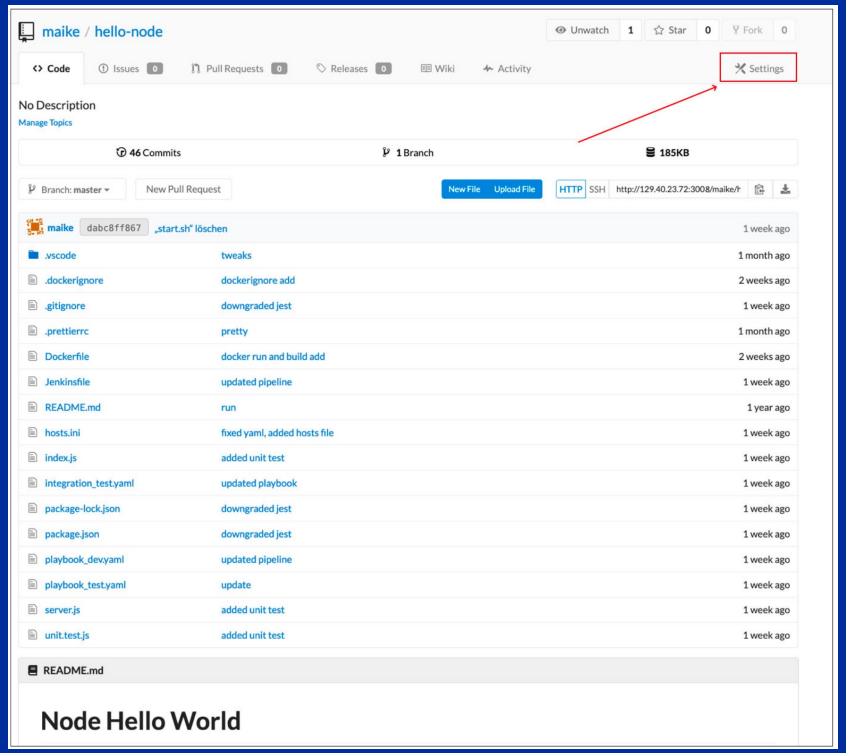
Jenkinsfile

```
pipeline {
    agent none
    stages {
        stage('Build') {
            agent{
                node {
                    label 'build'
                }
            }
            steps {
                checkout scm
                sh 'npm install'
                sh 'npm test'
                sh 'docker build -t localhost:5000/hello-node:latest .'
                sh 'docker push localhost:5000/hello-node'
            }
        }
        stage('Deploy') {
            agent{
                node {
                    label 'test'
                }
            }
            steps{
            ansiblePlaybook(inventory: 'hosts.ini', playbook: 'playbook_dev.yaml')
            ansiblePlaybook(inventory: 'hosts.ini', playbook: 'integration_test.yaml')
            ansiblePlaybook(inventory: 'hosts.ini', playbook: 'playbook_test.yaml')
            }
        }
    }
}
```

DevOps Use Case / Erstellen einer Pipeline

Webhook in Gitea erstellen

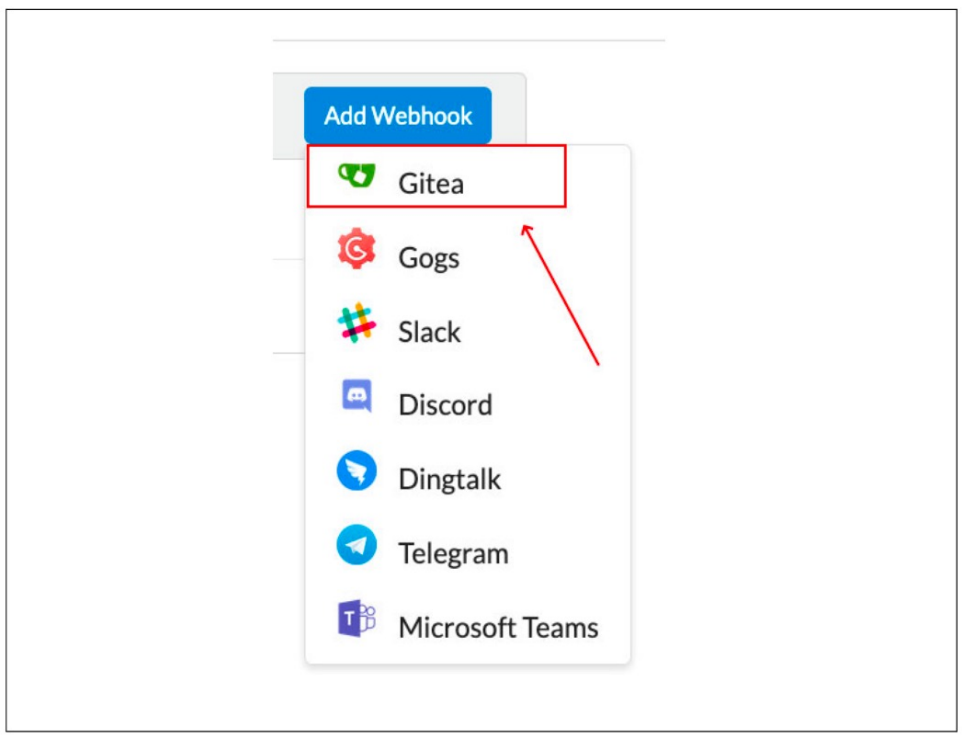
1. Im Gitea Repository, ins Einstellungen Menü



DevOps Use Case / Erstellen einer Pipeline

Webhook in Gitea erstellen

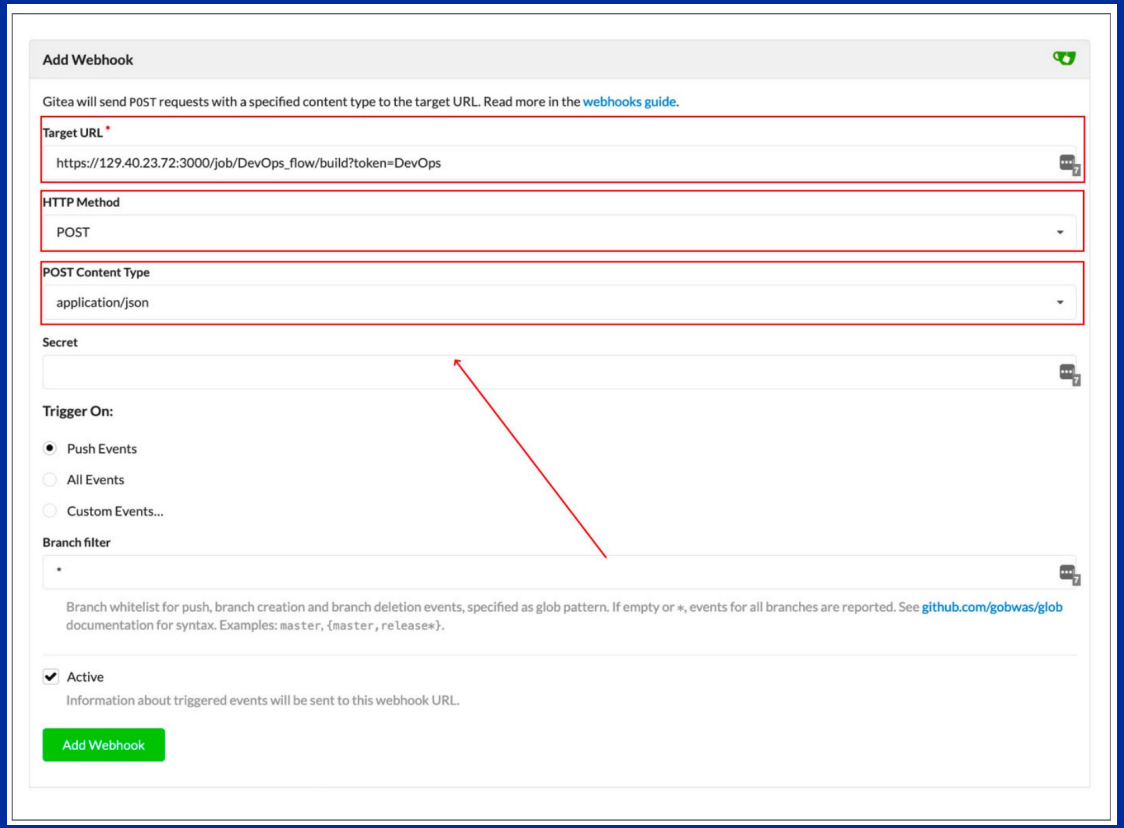
2. Anlegen eines Gitea Webhooks



DevOps Use Case / Erstellen einer Pipeline

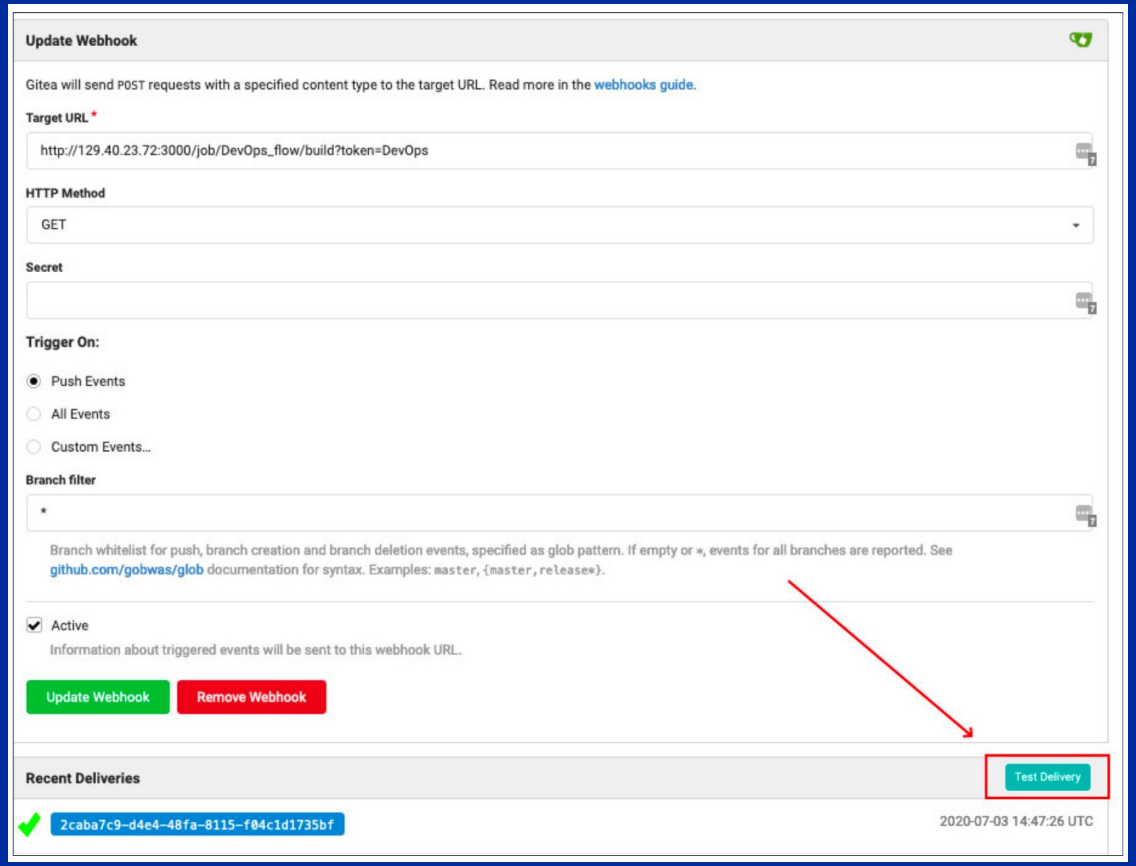
Webhook in Gitea erstellen

3. Webhook mit Jenkins Pipeline Trigger konfigurieren



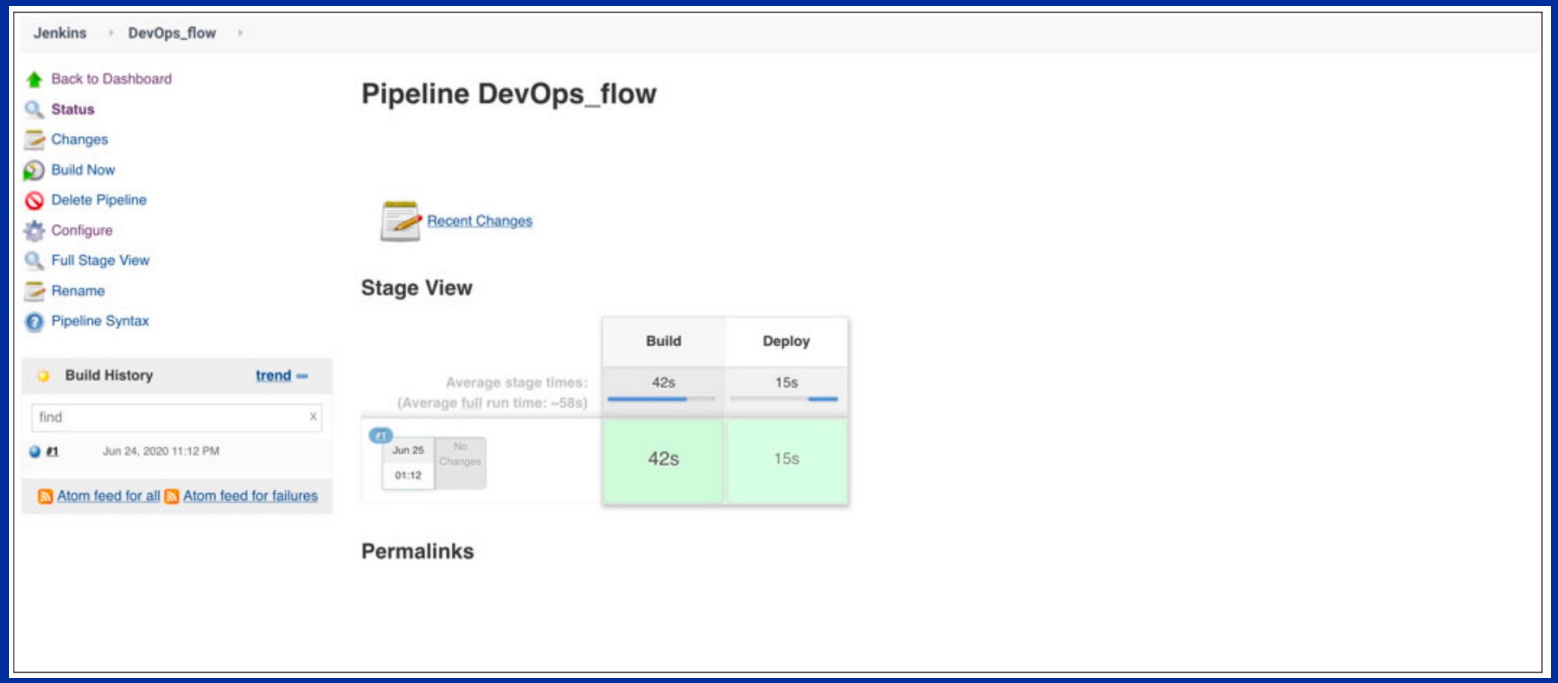
DevOps Use Case / Erstellen einer Pipeline

4. Pipeline starten



DevOps Use Case / Erstellen einer Pipeline

5. Im Jenkins Interface: erfolgreicher Pipeline Durchlauf



Praxis

MQ Gateway Client Concentrator

Szenario: Der Kunde betreibt MQ Server unter Windows/X86 und plant MQ in zCX um Kommunikation mit MQ z/OS zu vereinfachen sowie Target Que Manager und Concentrator zu co-locaten.

PoC: Der Kunde betreibt MQ Gateways in 2 identitischen Farmen, wovon eine in zCX umgezogen und vergleichsweise auf Performanz getestet wird.

Ergebnis:

	Windows	zCX @ Mainframe
Measured peak loads	25-28% (of total system)	15-17% (of defined zCX-Instance)
Measured minimum utilization	10% (of total system)	2,7% (of defined zCX-Instance)
Average workload	12-15%	5-7%
RAM-Consumption	constant > 3 GB (incl. Windows)	ca. 1,2 GB (incl. zCX/Docker)
Max. possible Messages (extrapolated)	13.200 Msg/h	28.700 Msg/h

Praxis

Service Management Unite

Szenario: Der Kunde betreibt SMU derzeit unter Linux in z/VM . Da er mit SMU zufrieden ist, möchte er dies nun auch für z/OS vertesten.

Projekt: Der Kunde ist nicht überzeugt von der Hochverfügbarkeit und dem persistenten Speichern von zCX Docker Containern (Derzeit nur 2 Möglichkeiten: Neustart (1 Min Outage oder permanent Speicher für eine neue zCX Instanz reservieren). Daher wurde ein continious availability Konzept auf Anwendungsebene am Beispiel von SMU verprobt.

Ergebnis: Mit Hilfe eines Load Balancers, DB2 Servers, 2 SMU zCX Environments und Event Dispatchers wurde ein continious availability Betriebskonzept erzielt.

Praxis

Watson Machine Learning for IBM Z

Szenario: Der Kunde hat aktuell das gesamte Trainining und Scoring seines Machine Learnings in der Cloud. Es werden Modelle trainiert und anschließend Bilder gescored/bewertet/ausgewertet. In Zukunft sollen allerdings Bilder mit einem hohen Schutzbedarf dazukommen (nicht Cloud).

PoC: Das Scoring der Machine Learning Applikation soll mit Hilfe von WMLz in zCX erfolgen. Die Bilder sollen dabei in DB2 auf z/OS gespeichert werden.

Ergebnis: Mit Hilfe einiger Anpassungen des WMLz Produkts konnten im PoC alle Modelle angewandt und Bilder gescored werden. Dies dauert wesentlich länger als in der Cloud, die Anwendung ist jedoch nicht zeitkritisch.

Praxis

Grafana

Szenario: Da zCX keinen Zugriff auf den Linux Kernel erlaubt monitored der Kunde seine zCX Umgebung mit Hilfe von Grafana.

Python Anwendungen

Szenario: Python Web Server einer Monitoring Applikation, die bisher unter RHEL 8 auf z/VM lief, erfolgreich umgezogen.

TPC-C OLTP Benchmark

Szenario: Der Kunde hat Applikation auf X86 die auf eine DB2 im z/OS zugreifen. Getestet wurden die möglichen Transaktionen pro Sekunde.
Ergebnis: zCX performte bei jedem Test mindestens 20% besser.

Danke

Maike Havemann
IBM Z Client Technical Specialist
—
maike.havemann@de.ibm.com
+49-174 1676826
ibm.com