|  |
| --- |
|  |

Setup Guide

Docker & Kubernetes

Draft

Author: Thorsten Illies

Version 1.0 | 28, 12, 2016

TABLE OF CONTENTS

[1 What is Kubernetes 3](#_Toc472256988)

[1.1 Why do I need Kubernetes and what can it do? 3](#_Toc472256989)

[1.2 Nodes 3](#_Toc472256990)

[1.3 Pods 3](#_Toc472256991)

[1.4 Replica Sets and Replication Controllers 4](#_Toc472256992)

[1.5 Services 4](#_Toc472256993)

[2 Kubernetes Setup 5](#_Toc472256994)

[2.1 Prerequisites 5](#_Toc472256995)

[2.2 Installing KUBELET and KUBEADM 5](#_Toc472256996)

[3 Create an Integration Server Container 6](#_Toc472256997)

[4 Running the Container in Kubernetes 7](#_Toc472256998)

[5 Useful commands 7](#_Toc472256999)

[5.1 Docker commands 7](#_Toc472257000)

[5.1.1 Docker images 7](#_Toc472257001)

[5.1.2 Docker run 7](#_Toc472257002)

[5.2 Kubernetes commands 8](#_Toc472257003)

[5.2.1 Kubectl run 8](#_Toc472257004)

[5.2.2 kubectl autoscale 8](#_Toc472257005)

[5.2.3 kubectl delete 9](#_Toc472257006)

# What is Kubernetes

The name **Kubernetes** originates from Greek, meaning “helmsman” or “pilot”, and is the root of “governor” and “cybernetic”. **K8s** is an abbreviation derived by replacing the 8 letters “ubernete” with 8.

Kubernetes is an [open-source platform for automating deployment, scaling, and operations of application containers](http://www.slideshare.net/BrianGrant11/wso2con-us-2015-kubernetes-a-platform-for-automating-deployment-scaling-and-operations) across clusters of hosts, providing container-centric infrastructure.

With Kubernetes, you are able to quickly and efficiently respond to customer demand:

* Deploy your applications quickly and predictably.
* Scale your applications on the fly.
* Seamlessly roll out new features.
* Optimize use of your hardware by using only the resources you need

## Why do I need Kubernetes and what can it do?

At a minimum, Kubernetes can schedule and run application containers on clusters of physical or virtual machines. However, Kubernetes also allows developers to ‘cut the cord’ to physical and virtual machines, moving from a **host-centric** infrastructure to a **container-centric** infrastructure, which provides the full advantages and benefits inherent to containers. Kubernetes provides the infrastructure to build a truly **container-centric** development environment.

## Nodes

A node is a worker machine in Kubernetes, previously known as a minion. A node may be a VM or physical machine, depending on the cluster. Each node has the services necessary to run pods and is managed by the master components. The services on a node include Docker, kubelet and kube-proxy. See The Kubernetes Node section in the architecture design doc for more details.

## Pods

Pods are the smallest deployable units of computing that can be created and managed in Kubernetes. This sometimes leads to the confusion that pods are single containers, as that's what people are used to from Docker. While pods can contain one single container, they are not limited to one and can contain as many containers as needed.

What makes these containers a pod, is that all containers in a pod run as if they would have been running on a single host in pre-container world. Thus, they share a set of Linux namespaces and do not run isolated from each other. This results in them sharing an IP address and port space, and being able to find each other over localhost or communicate over the IPC namespace. Further, all containers in a pod have access to shared volumes, that is they can mount and work on the same volumes if needed.

In order to gain all this functionality a pod is a single deployable unit. Each single instance of the pod (with all its containers) is always scheduled together.

Now for the typical Docker user this concept is quite new. While Giant Swarm users have been able to use pods for quite a while even without Kubernetes, only few really do. For some it might sound like going back from the isolated "one process per container" to "deploying your whole LAMP stack together". However, this is not the intended use case for pods. The main motivation for the pod concept is supporting co-located, co-managed helper containers next to the application container. These include things like: logging or monitoring agents, backup tooling, data change watchers, event publishers, proxies, etc. If you are not sure what to use pods for in the beginning, you can for now use them with single containers like you might be used to from Docker.

The pod is the most basic concept in Kubernetes. By itself, it is ephemeral and won't be rescheduled to a new node once it dies. If we want to keep one or more instances of a pod alive we need another concept: replica sets. But before that we need to understand what labels and selectors are.

## Replica Sets and Replication Controllers

Replica sets, for those who have read a bit about Kubernetes before, are the next-generation of replication controllers. Currently, the main difference being that replica sets support the more advanced set-based selectors and thus are more flexible than replication controllers. However, the gist of the following explanation fits to both.

As mentioned above a pod by itself is ephemeral and won't be rescheduled if the node it is running on goes down. This is where the replica set comes in and ensures that a specific number of pod instances (or *replicas*) are running at any given time. Thus, if you want your pod to stay alive you make sure you have an according replica set specifying at least one replica for that pod. The replica set then takes care of (re)scheduling your instances for you.

As indicated above the replica set ensures a specific number of replicas are running. By modifying the number of replicas in the set's definition you can scale your pods up and down.

A replica set can not only manage a single pod but also a group of different pods selected based on a common label. This enables a replica set to for example scale all pods that together compose the frontend of an application together without having to have identical replica sets for each pod in the frontend.

You can include the definition of the pod directly in the definition of the replica set, so you can manage them together. However, there is a higher level concept called a *deployment*, which manages replica sets. Therefore, you usually won't need to create or manipulate replica set objects directly. It's still important knowing about this concept as without it you won't understand the specific workings of how Kubernetes will help you run and manage your applications. I will explain deployments and the many features they bring with them in a follow-up blog post, for now suffice to say that they will manage your replica sets for you.

## Services

Kubernetes Pods are mortal. They are born and when they die, they are not resurrected. [ReplicationControllers](https://kubernetes.io/docs/user-guide/replication-controller) in particular create and destroy Pods dynamically (e.g. when scaling up or down or when doing rolling updates). While each Pod gets its own IP address, even those IP addresses cannot be relied upon to be stable over time. This leads to a problem: if some set of Pods (let’s call them backends) provides functionality to other Pods (let’s call them frontends) inside the Kubernetes cluster, how do those frontends find out and keep track of which backends are in that set?

Enter Services. A Kubernetes Service is an abstraction which defines a logical set of Pods and a policy by which to access them - sometimes called a micro-service. The set of Pods targeted by a Service is (usually) determined by a Label Selector (see below for why you might want a Service without a selector).

As an example, consider an image-processing backend which is running with 3 replicas. Those replicas are fungible - frontends do not care which backend they use. While the actual Pods that compose the backend set may change, the frontend clients should not need to be aware of that or keep track of the list of backends themselves. The Service abstraction enables this decoupling.

For Kubernetes-native applications, Kubernetes offers a simple Endpoints API that is updated whenever the set of Pods in a Service changes. For non-native applications, Kubernetes offers a virtual-IP-based bridge to Services which redirects to the backend Pods.

# Kubernetes Setup

## Prerequisites

1. One or more machines running CentOS 7
2. 1GB or more of RAM
3. Full network connectivity between all machines in the cluster (public or private network is fine)

## Installing KUBELET and KUBEADM

You will install the following packages on all the machines:

* **docker**: the container runtime, which Kubernetes depends on.
* **kubelet**: the most core component of Kubernetes. It runs on all of the machines in your cluster and does things like starting pods and containers.
* **kubectl**: the command to control the cluster once it’s running. You will only need this on the master, but it can be useful to have on the other nodes as well.
* **kubeadm**: the command to bootstrap the cluster.

Instructions:

1. SSH into the machine and become root if you are not already (for example, run sudo su -).
2. Enter:  
   cat <<EOF > /etc/yum.repos.d/kubernetes.repo  
   [kubernetes]  
   name=Kubernetes  
   baseurl=http://yum.kubernetes.io/repos/kubernetes-el7-x86\_64  
   enabled=1  
   gpgcheck=1  
   repo\_gpgcheck=1  
   gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg  
    <https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg>  
   EOF  
   setenforce 0  
   yum install -y docker kubelet kubeadm kubectl kubernetes-cni  
   systemctl enable docker && systemctl start docker  
   systemctl enable kubelet && systemctl start kubelet
3. Enter:  
   setenforce 0
4. Initialize your Master by running:   
   kubeadm init
5. Make a record of the *kubeadm join* command that *kubeadm init* returns.
6. If you want to schedule Pods on the Master do: kubectl taint nodes --all dedicated-
7. To install a Pod Network you can use e.g. Weaves: kubectl apply -f https://git.io/weave-kube
8. To Check, that all Pods are up and running, you cann issue the following command: kubectl get pods --all-namespaces

After this, your Kubernetes Cluster is setup and ready for the deployment of Pods.

# Create an Integration Server Container

Copy following file as Dockerfile\_IS under your SoftwareAG installation directory such as /opt/SoftwareAG:

FROM centos:7

MAINTAINER SoftwareAG

COPY ./jvm/jvm/ /sag/jvm/jvm/

ENV JAVA\_HOME /sag/jvm/jvm

COPY ./install/jars/ /sag/install/jars/

#Comment the line below for 9.7 version of Integration Server

COPY ./install/profile/ /sag/install/profile/

COPY ./install/products/ /sag/install/products/

COPY ./Licenses/sagosch /sag/Licenses/sagosch

COPY ./common/bin/ /sag/common/bin/

COPY ./common/conf/ /sag/common/conf/

COPY ./common/lib/ /sag/common/lib/

COPY ./common/runtime/ /sag/common/runtime/

COPY ./common/security/ /sag/common/security/

COPY ./WS-Stack/ /sag/WS-Stack/

COPY ./IntegrationServer/bin/ /sag/IntegrationServer/bin/

COPY ./IntegrationServer/lib/ /sag/IntegrationServer/lib/

COPY ./IntegrationServer/sdk/ /sag/IntegrationServer/sdk/

COPY ./IntegrationServer/updates/ /sag/IntegrationServer/updates/

COPY ./IntegrationServer/web/ /sag/IntegrationServer/web/

COPY ./IntegrationServer/.tc.dev.log4j.properties /sag/IntegrationServer/.tc.dev.log4j.properties

COPY ./IntegrationServer/instances/default/ /sag/IntegrationServer/instances/default/

COPY ./IntegrationServer/instances/lib/ /sag/IntegrationServer/instances/lib/

COPY ./IntegrationServer/instances/is\_instance.xml /sag/IntegrationServer/instances/is\_instance.xml

COPY ./IntegrationServer/instances/is\_instance.sh /sag/IntegrationServer/instances/is\_instance.sh

COPY ./profiles/IS\_default/configuration/custom\_wrapper.conf /sag/IntegrationServer/instances/custom\_wrapper.conf.template

RUN cd /sag/IntegrationServer/instances; ./is\_instance.sh updateServerCnfFile -Dinstance.name=default

RUN cd /sag/IntegrationServer/instances; ./is\_instance.sh create-osgi-profile -Dinstance.name=default

CMD cd /sag/profiles/IS\_default/bin && ./console.sh

EXPOSE 5555

EXPOSE 9999

Execute the following commands:

cd /opt/SoftwareAG ==> Change Directory to SoftwareAG installation directory where you copied Dockerfile\_IS

docker build -t <imageName> -f Dockerfile\_IS .

List all the docker images and there you should find the newly created image:

docker images

To run your newly build container: docker run -d --name IS\_Default -p 5555 -p 9999 <imageName>

# Running the Container in Kubernetes

To run your container under Kubernetes you simply need to execute

kubectl run <name> --image=<image> --replicas=<number of replicas>

# Useful commands

## Docker commands

### Docker images

Usage: docker images [OPTIONS] [REPOSITORY[:TAG]]

List images

Options:

-a, --all Show all images (default hides intermediate images)

--digests Show digests

-f, --filter value Filter output based on conditions provided (default [])

- dangling=(true|false)

- label=<key> or label=<key>=<value>

- before=(<image-name>[:tag]|<image-id>|<image@digest>)

- since=(<image-name>[:tag]|<image-id>|<image@digest>)

--format string Pretty-print images using a Go template

--help Print usage

--no-trunc Don't truncate output

-q, --quiet Only show numeric IDs

### Docker run

$ docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]

docker run -i -t wmicrepo:is912\_base /bin/bash

## Kubernetes commands

### Kubectl run

Here are some sample kubectl run commands:

# Start a single instance of nginx.

kubectl run nginx --image=nginx

# Start a single instance of <name> and let the container expose port 5701 .

kubectl run <name> --image=<name> --port=5701

# Start a single instance of hazelcast and set environment variables "DNS\_DOMAIN=cluster" and "POD\_NAMESPACE=default" in the container.

kubectl run <name> --image=<name> --env="DNS\_DOMAIN=cluster" --env="POD\_NAMESPACE=default"

# Start a replicated instance of nginx.

kubectl run <name> --image=<name> --replicas=5

# Dry run. Print the corresponding API objects without creating them.

kubectl run <name> --image=<name> --dry-run

# Start a single instance of nginx, but overload the spec of the deployment with a partial set of values parsed from JSON.

kubectl run <name> --image=<name> --overrides='{ "apiVersion": "v1", "spec": { ... } }'

# Start a pod of busybox and keep it in the foreground, don't restart it if it exits.

kubectl run -i -t busybox --image=busybox --restart=Never

# Start the nginx container using the default command, but use custom arguments (arg1 .. argN) for that command.

kubectl run nginx --image=nginx -- <arg1> <arg2> ... <argN>

# Start the nginx container using a different command and custom arguments.

kubectl run nginx --image=nginx --command -- <cmd> <arg1> ... <argN>

# Start the perl container to compute π to 2000 places and print it out.

kubectl run pi --image=perl --restart=OnFailure -- perl -Mbignum=bpi -wle 'print bpi(2000)'

# Start the cron job to compute π to 2000 places and print it out every 5 minutes.

kubectl run pi --schedule="0/5 \* \* \* ?" --image=perl --restart=OnFailure -- perl -Mbignum=bpi -wle 'print bpi(2000)'

### kubectl autoscale

# Auto scale a deployment "foo", with the number of pods between 2 and 10, target CPU utilization specified so a default autoscaling policy will be used:

kubectl autoscale deployment foo --min=2 --max=10

# Auto scale a replication controller "foo", with the number of pods between 1 and 5, target CPU utilization at 80%:

kubectl autoscale rc foo --max=5 --cpu-percent=80

### kubectl delete

# Delete a pod using the type and name specified in pod.json.

kubectl delete -f ./pod.json

# Delete a pod based on the type and name in the JSON passed into stdin.

cat pod.json | kubectl delete -f -

# Delete pods and services with same names "baz" and "foo"

kubectl delete pod,service baz foo

# Delete pods and services with label name=myLabel.

kubectl delete pods,services -l name=myLabel

# Delete a pod with minimal delay

kubectl delete pod foo --now

# Force delete a pod on a dead node

kubectl delete pod foo --grace-period=0 --force

# Delete a pod with UID 1234-56-7890-234234-456456.

kubectl delete pod 1234-56-7890-234234-456456

# Delete all pods

kubectl delete pods --all

|  |
| --- |
| ABOUT SOFTWARE AG  Software AG offers the world’s first Digital Business Platform. Recognized as a leader by the industry’s top analyst firms, Software AG helps you combine existing systems on premises and in the cloud into a single platform to optimize your business and delight your customers. With Software AG, you can rapidly build and deploy Digital Business Applications to exploit real-time market opportunities. Get maximum value from big data, make better decisions with streaming analytics, achieve more with the Internet of Things, and respond faster to shifting regulations and threats with intelligent governance, risk and compliance. The world’s top brands trust Software AG to help them rapidly innovate, differentiate and win in the digital world. Learn more at www.SoftwareAG.com.  © 2016 Software AG. All rights reserved. Software AG and all Software AG products are either trademarks or registered trademarks of Software AG. Other product and company names mentioned herein may be the trademarks of their respective owners |
|  |