# Early delivery of the CKAF docker images

# Introduction

Docker images for KAFKA are supported since CKAF 17.6. We've included an overview of image design, a quickstart guide and advanced topics.

Since CKAF 17.3, we deliver the software packages integrated by Confluent platform, such as Zookeeper, Kafka, Schema-registry and REST proxy. When building the docker images, we have referred to the open source images from Confluentinc and reused their code.

# Image design

There is a basic docker image built from REHL for all the CSF components. All the software packages the images need are downloaded from the internal repos and the private nexus website.

For early delivery, we only supply ckaf-zookeeper and ckaf-Kafka. These two images are available on nexus

There are some useful scripts to help to generate config files, such as server.properties and log4j.properties, and to ensure the process is started normally.

The docker images are built by docker 1.12.6.

The tag of the two images are both v1, that is, ckaf-zookeeper:v1 and ckaf-kafka:v1.

# Quickstart

This section provides a basic guide for deploying a Kafka cluster along in your docker environment. In order to keep things simple, we'll start with a single node docker environment. We will also be configuring Kafka and Zookeeper to store data locally in their docker containers.

The way to deploy a Kafka cluster will be mentioned later.

1. Examples of how to add mounted volumes to your host machines are available later in this documentation.
2. To get started, you'll need to first install Docker and get it running. The Docker Images require Docker version 1.11 or greater.

## Install docker and download the images

Try to run docker info in your host. If docker is not installed, install it as below.

### Config yum repo

Following yum repo can be used, add it into /etc/yum.repos.d/

# cat /etc/yum.repos.d/nokia.repo

[rhel‐x86\_64‐server‐7]

name="rhel‐x86\_64‐server‐7"

baseurl=http://lss‐repo1.ih.lucent.com/repos/7/latest/dist/rhel/server/7/7Serve

r/x86\_64/os/

enabled=1

gpgcheck=0

sslverify=0

[rhel‐x86\_64‐server‐extras‐7]

name="rhel‐x86\_64‐server‐extras‐7"

baseurl=http://lss‐repo1.ih.lucent.com/repos/7/latest/dist/rhel/server/7/7Serve

r/x86\_64/extras/os/

enabled=1

gpgcheck=0

sslverify=0

### Run yum install

yum install docker

### Start docker server

After the package is installed, you can start the docker server by

systemctl start docker

This command must be run as root.

The docker server will start with the default configurations.

### Download the images

For early delivery, the images are stored on nexus, you could download them by:

wget <http://repository.app.alcatel-lucent.com/nexus/service/local/repositories/CSF-NOSRC/content/Drafts/CKAF/Bundles/ckaf-zookeeper/0.1/ckaf-zookeeper-0.1.tar>

and

wget <http://repository.app.alcatel-lucent.com/nexus/service/local/repositories/CSF-NOSRC/content/Drafts/CKAF/Bundles/ckaf-kafka/0.1/ckaf-kafka-0.1.tar>

After the download is completed, load the packages by:

docker load --input ckaf-zookeeper-0.1.tar

docker load --input ckaf-kafka-0.1.tar

After that, you can see the images are loaded by docker images if no error happens.

The output is like this:

[root@localhost ainet]# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

ckaf-kafka v1 4cbbf2b4df9b 8 days ago 396.2 MB

ckaf-zookeeper v1 2b1f3f3d7e6c 8 days ago 396.2 MB

Until now, the docker images are ready to use.

## Start zookeeper

The Zookeeper image uses variables prefixed with ZOOKEEPER\_ with the variables expressed exactly as they would appear in the zookeeper.properties file. As an example, to set clientPort, tickTime, and syncLimit run the command below:

docker run -d \

--net=host \

--name=zookeeper \

-e ZOOKEEPER\_CLIENT\_PORT=32181 \

-e ZOOKEEPER\_TICK\_TIME=2000 \

-e ZOOKEEPER\_SYNC\_LIMIT=2

ckaf-zookeeper:v1

Required Settings:

|  |  |
| --- | --- |
| ZOOKEEPER\_CLIENT\_PORT | This field is always required for telling zookeeper where to listen for connections by clients such as Kafka. |
| ZOOKEEPER\_SERVER\_ID | Only required when running in clustered mode. Set the server ID in the myid file, which consists of a single line containing only the text of that machine's id. So myid of server 1 would contain the text "1" and nothing else. The id must be unique within the ensemble and should have a value between 1 and 255. |

You'll need to keep this service running throughout, so use a dedicated terminal window if you plan to launch the image in the foreground.

docker run -d \

--net=host \

--name=zookeeper \

-e ZOOKEEPER\_CLIENT\_PORT=32181 \

ckaf-zookeeper:v1

This command instructs docker to launch an instance of the ckaf-zookeeper container and name it zookeeper. We also specify that we want to use host networking and pass in the required parameter for running Zookeeper: ZOOKEEPER\_CLIENT\_PORT.

We'll check the docker logs to confirm that the container has booted up successfully and started the Zookeeper service. The command to do that is:

docker logs zookeeper

With this command, we're referencing the container name we want to see the logs for. To list all containers (running or failed), you can always run docker ps -a. This is especially useful when running in detached mode.

When you output the logs for Zookeeper, you should see the following message at the end of the log output:

[2016-07-24 05:15:35,453] INFO binding to port 0.0.0.0/0.0.0.0:32181 (org.apache.zookeeper.server.NIOServerCnxnFactory)

Note that the message shows the Zookeeper service listening at the port we passed in as ZOOKEEPER\_CLIENT\_PORT above. Because we are using the host network mode that docker provides, the process is listening on port 32181 of the docker host. You can run the command to check it.

netstat –anp|grep 32181

If the service is not running, the log messages should provide details to help you identify the problem. Some common errors include:

Network port already in use

In that case, you'll see a message indicating that the Zookeeper service could not bind to the selected port. Simply change to an open port or identify (and stop) the Docker container that has a service using that port.

Insufficient resources

In rare occasions, you may see memory allocation or other low-level failures at startup. This will only happen if you dramatically overload the capacity of your Docker host.

## Start Kafka

The Kafka image uses variables prefixed with KAFKA\_ with an underscore (\_) separating each word instead of periods. As an example, to set broker.id, advertised.listeners and zookeeper.connect you'd run the following command:

docker run -d \

--net=host \

--name=kafka \

-e KAFKA\_ZOOKEEPER\_CONNECT=localhost:32181 \

-e KAFKA\_ADVERTISED\_LISTENERS=PLAINTEXT://localhost:29092 \

-e KAFKA\_BROKER\_ID=2 \

ckaf-kafka:v1

Note:

You'll notice that we set the KAFKA\_ADVERTISED\_LISTENERS variable to localhost:29092. This is an important setting, as it will make Kafka accessible from outside the container by advertising its location on the docker host.

Required Settings:

|  |  |
| --- | --- |
| KAFKA\_ZOOKEEPER\_CONNECT | Tell Kafka how to get in touch with zookeeper. |
| KAFKA\_ADVERTISED\_LISTENERS | Advertised listeners is required for starting up the Docker image because it is important to think through how other clients are going to connect to Kafka. In a Docker environment, you will need to make sure that your clients can connect to Kafka and other services. Advertised.listeners is for how it gives out a host name that can be reached by the client. |

Run the following command to start the Kafka process in a container named “kafka”.

docker run -d \

--net=host \

--name=kafka \

-e KAFKA\_ZOOKEEPER\_CONNECT=localhost:32181 \

-e KAFKA\_ADVERTISED\_LISTENERS=PLAINTEXT://localhost:29092 \

ckaf-kafka:v1

You'll notice that we set the KAFKA\_ADVERTISED\_LISTENERS variable to localhost:29092. This will make Kafka accessible from outside the container by advertising its location on the Docker host. We also passed in the zookeeper port we used when launching that container a moment ago. Because we are using --net=host, the hostname for the zookeeper service can be left at localhost.

Let's check the logs to see the broker has booted up successfully:

docker logs kafka

You should see the following at the end of the log output:

....

[2016-07-15 23:31:00,295] INFO [Kafka Server 1], started (kafka.server.KafkaServer)

[2016-07-15 23:31:00,295] INFO [Kafka Server 1], started (kafka.server.KafkaServer)

...

...

[2016-07-15 23:31:00,349] INFO [Controller 1]: New broker startup callback for 1 (kafka.controller.KafkaController)

[2016-07-15 23:31:00,349] INFO [Controller 1]: New broker startup callback for 1 (kafka.controller.KafkaController)

[2016-07-15 23:31:00,350] INFO [Controller-1-to-broker-1-send-thread], Starting (kafka.controller.RequestSendThread)

...

## Basic user scenario

Now we can take this very basic deployment for a test drive. We'll verify that the broker is functioning normally by creating a topic and producing data to it. We'll use the client tools directly from another Docker container.

First, we'll create a topic. We'll name it foo and keep things simple by just giving it one partition and only one replica. Production environments with more broker nodes would obviously use higher values for both partitions and replicas for scalability and resiliency.

docker run --net=host --rm ckaf-kafka:v1 \

kafka-topics --create --topic foo --partitions 1 --replication-factor 1 --if-not-exists --zookeeper localhost:32181

You should see the following output in your terminal window:

Created topic "foo"

Before moving on, verify that the topic was created successfully:

docker run --net=host --rm ckaf-kafka:v1 \

kafka-topics --describe --topic foo --zookeeper localhost:32181

The response should be:

Topic:foo PartitionCount:1 ReplicationFactor:1 Configs:

Topic: foo Partition: 0 Leader: 1001 Replicas: 1001 Isr: 1001

Next, we'll publish some data to our new topic:

docker run --net=host --rm ckaf-kafka:v1 \

bash -c "seq 42 | kafka-console-producer --request-required-acks 1 --broker-list localhost:29092 --topic foo && echo 'Produced 42 messages.'"

This command will use the built-in Kafka Console Producer to produce 42 simple messages to the topic. Upon running it, you should see the following:

Produced 42 messages.

To complete the story, let's read back the message using the built-in Console consumer:

docker run --net=host --rm ckaf-kafka:v1\

kafka-console-consumer --bootstrap-server localhost:29092 --topic foo --new-consumer --from-beginning --max-messages 42

If everything is working as expected, each of the original messages we produced should be written back out:

1

....

42

Processed a total of 42 messages

## Clustered Deployment

In this section, we provide a tutorial for running a three-node Kafka cluster and Zookeeper ensemble in one docker host.

### Start Up a 3-node Zookeeper Ensemble

To start zookeeper node 1:

docker run -d \

--net=host \

--name=zk-1 \

-e ZOOKEEPER\_SERVER\_ID=1 \

-e ZOOKEEPER\_CLIENT\_PORT=22181 \

-e ZOOKEEPER\_TICK\_TIME=2000 \

-e ZOOKEEPER\_INIT\_LIMIT=5 \

-e ZOOKEEPER\_SYNC\_LIMIT=2 \

-e ZOOKEEPER\_SERVERS="localhost:22888:23888;localhost:32888:33888;localhost:42888:43888" \

ckaf-zookeeper:v1

To start zookeeper node 2:

docker run -d \

--net=host \

--name=zk-2 \

-e ZOOKEEPER\_SERVER\_ID=2 \

-e ZOOKEEPER\_CLIENT\_PORT=32181 \

-e ZOOKEEPER\_TICK\_TIME=2000 \

-e ZOOKEEPER\_INIT\_LIMIT=5 \

-e ZOOKEEPER\_SYNC\_LIMIT=2 \

-e ZOOKEEPER\_SERVERS="localhost:22888:23888;localhost:32888:33888;localhost:42888:43888" \

ckaf-zookeeper:v1

To start zookeeper node 3:

docker run -d \

--net=host \

--name=zk-3 \

-e ZOOKEEPER\_SERVER\_ID=3 \

-e ZOOKEEPER\_CLIENT\_PORT=42181 \

-e ZOOKEEPER\_TICK\_TIME=2000 \

-e ZOOKEEPER\_INIT\_LIMIT=5 \

-e ZOOKEEPER\_SYNC\_LIMIT=2 \

-e ZOOKEEPER\_SERVERS="localhost:22888:23888;localhost:32888:33888;localhost:42888:43888" \

ckaf-zookeeper:v1

Before moving on, we'll check the logs to see the broker has booted up successfully by running the following command:

docker logs zk-1

You should see messages like this at the end of the log output:

[2016-07-24 07:17:50,960] INFO Created server with tickTime 2000 minSessionTimeout 4000 maxSessionTimeout 40000 datadir /var/lib/zookeeper/log/version-2 snapdir /var/lib/zookeeper/data/version-2 (org.apache.zookeeper.server.ZooKeeperServer)

[2016-07-24 07:17:50,961] INFO FOLLOWING - LEADER ELECTION TOOK - 21823 (org.apache.zookeeper.server.quorum.Learner)

[2016-07-24 07:17:50,983] INFO Getting a diff from the leader 0x0 (org.apache.zookeeper.server.quorum.Learner)

[2016-07-24 07:17:50,986] INFO Snapshotting: 0x0 to /var/lib/zookeeper/data/version-2/snapshot.0 (org.apache.zookeeper.server.persistence.FileTxnSnapLog)

[2016-07-24 07:17:52,803] INFO Received connection request /127.0.0.1:50056 (org.apache.zookeeper.server.quorum.QuorumCnxManager)

[2016-07-24 07:17:52,806] INFO Notification: 1 (message format version), 3 (n.leader), 0x0 (n.zxid), 0x1 (n.round), LOOKING (n.state), 3 (n.sid), 0x0 (n.peerEpoch) FOLLOWING (my state) (org.apache.zookeeper.server.quorum.FastLeaderElection)

You can repeat the command for the two other Zookeeper nodes. Next, you should verify that ZK ensemble is ready:

for i in 22181 32181 42181; do

docker run --net=host --rm ckaf-zookeeper:v1 bash -c "echo stat | nc localhost $i | grep Mode"

done

You should see one leader and two follower nodes. The output should look something like the following:

Mode: follower

Mode: leader

Mode: follower

### Set up a three node Kafka cluster

Now that Zookeeper is up and running, we can fire up a three node Kafka cluster.

To start Kafka node 1:

docker run -d \

--net=host \

--name=kafka-1 \

-e KAFKA\_ZOOKEEPER\_CONNECT=localhost:22181,localhost:32181,localhost:42181 \

-e KAFKA\_ADVERTISED\_LISTENERS=PLAINTEXT://localhost:29092 \

ckaf-kafka:v1

To start Kafka node 2:

docker run -d \

--net=host \

--name=kafka-2 \

-e KAFKA\_ZOOKEEPER\_CONNECT=localhost:22181,localhost:32181,localhost:42181 \

-e KAFKA\_ADVERTISED\_LISTENERS=PLAINTEXT://localhost:39092 \

ckaf-kafka:v1

To start Kafka node 3:

docker run -d \

--net=host \

--name=kafka-3 \

-e KAFKA\_ZOOKEEPER\_CONNECT=localhost:22181,localhost:32181,localhost:42181 \

-e KAFKA\_ADVERTISED\_LISTENERS=PLAINTEXT://localhost:49092 \

ckaf-kafka:v1

Check the logs to see the broker has booted up successfully

docker logs kafka-1

docker logs kafka-2

docker logs kafka-3

You should see start see bootup messages. For example, docker logs kafka-3 | grep started will show the following:

[2016-07-24 07:29:20,258] INFO [Kafka Server 1003], started (kafka.server.KafkaServer)

[2016-07-24 07:29:20,258] INFO [Kafka Server 1003], started (kafka.server.KafkaServer)

You should see the messages like the following on the broker acting as controller.

[2016-07-24 07:29:20,283] TRACE Controller 1001 epoch 1 received response {error\_code=0} for a request sent to broker localhost:29092 (id: 1001 rack: null) (state.change.logger)

[2016-07-24 07:29:20,283] TRACE Controller 1001 epoch 1 received response {error\_code=0} for a request sent to broker localhost:29092 (id: 1001 rack: null) (state.change.logger)

[2016-07-24 07:29:20,286] INFO [Controller-1001-to-broker-1003-send-thread], Starting (kafka.controller.RequestSendThread)

[2016-07-24 07:29:20,286] INFO [Controller-1001-to-broker-1003-send-thread], Starting (kafka.controller.RequestSendThread)

[2016-07-24 07:29:20,286] INFO [Controller-1001-to-broker-1003-send-thread], Starting (kafka.controller.RequestSendThread)

[2016-07-24 07:29:20,287] INFO [Controller-1001-to-broker-1003-send-thread], Controller 1001 connected to localhost:49092 (id: 1003 rack: null) for sending state change requests (kafka.controller.RequestSendThread)

### Basic operations

Test that the broker is working as expected.

Now that the brokers are up, we'll test that they're working as expected by creating a topic.

docker run --net=host --rm ckaf-kafka:v1 \

kafka-topics --create --topic bar --partitions 3 --replication-factor 3 --if-not-exists --zookeeper localhost:32181

You should see the following output:

Created topic "bar".

Now verify that the topic is created successfully by describing the topic.

docker run --net=host --rm ckaf-kafka:v1 \

kafka-topics --describe --topic bar --zookeeper localhost:32181

You should see the following message in your terminal window:

Topic:bar PartitionCount:3 ReplicationFactor:3 Configs:

Topic: bar Partition: 0 Leader: 1003 Replicas: 1003,1002,1001 Isr: 1003,1002,1001

Topic: bar Partition: 1 Leader: 1001 Replicas: 1001,1003,1002 Isr: 1001,1003,1002

Topic: bar Partition: 2 Leader: 1002 Replicas: 1002,1001,1003 Isr: 1002,1001,1003

Next, we'll try generating some data to the bar topic we just created.

docker run --net=host --rm ckaf-kafka:v1 \

bash -c "seq 42 | kafka-console-producer --broker-list localhost:29092 --topic bar && echo 'Produced 42 messages.'"

The command above will pass 42 integers using the Console Producer that is shipped with Kafka. As a result, you should see something like this in your terminal:

Produced 42 messages.

It looked like things were successfully written, but let's try reading the messages back using the Console Consumer and make sure they're all accounted for.

docker run --net=host --rm ckaf-kafka:v1 \

kafka-console-consumer --bootstrap-server localhost:29092 --topic bar --new-consumer --from-beginning --max-messages 42

You should see the following (it might take some time for this command to return data. Kafka has to create the \_\_consumers\_offset topic behind the scenes when you consume data for the first time and this may take some time):

1

4

7

10

13

16

....

41

Processed a total of 42 messages.

# Advanced topics

When we are setting Kafka in docker environment, there are many points to think about, such as persisting data, network configuration and security consideration.

The early delivery does not cover security.

## Available configurations

All the configs are passed by setting the environment variables.

Here is a list of all the configs of ckaf-zookeeper:

|  |  |
| --- | --- |
| **ENV names** | **Real config names** |
| ZOOKEEPER\_TICK\_TIME | tickTime |
| ZOOKEEPER\_GLOBAL\_OUTSTANDING\_LIMIT | globalOutstandingLimit |
| ZOOKEEPER\_PRE\_ALLOC\_SIZE | preAllocSize |
| ZOOKEEPER\_SNAP\_COUNT | snapCount |
| ZOOKEEPER\_TRACE\_FILE | traceFile |
| ZOOKEEPER\_MAX\_CLIENT\_CNXNS | maxClientCnxns |
| ZOOKEEPER\_CLIENT\_PORT\_ADDRESS | clientPortAddress |
| ZOOKEEPER\_MIN\_SESSION\_TIMEOUT | minSessionTimeout |
| ZOOKEEPER\_MAX\_SESSION\_TIMEOUT | maxSessionTimeout |
| ZOOKEEPER\_FSYNC\_WARNING\_THRESHOLDMS | fsync.warningthresholdms |
| ZOOKEEPER\_AUTOPURGE\_SNAP\_RETAIN\_COUNT | autopurge.snapRetainCount |
| ZOOKEEPER\_AUTOPURGE\_PURGE\_INTERVAL | autopurge.purgeInterval |
| ZOOKEEPER\_SYNC\_ENABLED | syncEnabled |
| ZOOKEEPER\_ELECTION\_ALG | electionAlg |
| ZOOKEEPER\_INIT\_LIMIT | initLimit |
| ZOOKEEPER\_LEADER\_SERVES | leaderServes |
| ZOOKEEPER\_SYNC\_LIMIT | syncLimit |
| ZOOKEEPER\_CNX\_TIMEOUT | cnxTimeout |
| ZOOKEEPER\_FORCE\_SYNC | forceSync |
| ZOOKEEPER\_JUTE\_MAX\_BUFFER | jute.maxbuffer |

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

|  |  |
| --- | --- |
| ZOOKEEPER\_SKIP\_ACL | skipACL |
| ZOOKEEPER\_QUORUM\_LISTEN\_ON\_ALL\_IPS | quorumListenOnAllIPs |

Once set, the configs will be generated in the zookeeer.properties and read by the script to start the zookeeper process.

For ckaf-kafka, the names of the options to start Kafka process are fixed. They are

|  |  |
| --- | --- |
| KAFKA\_VERSION | Options for the version of Kafka |
| KAFKA\_HEAP\_OPTS | Options for JVM heap |
| KAFKA\_LOG4J\_OPTS | Options for logging |
| KAFKA\_OPTS | Options for generic JVM settings you want to add |
| KAFKA\_JMX\_OPTS | Options about JMX |
| KAFKA\_JVM\_PERFORMANCE\_OPTS | Options about setting JVM performance |

For other Kafka broker configs, such as delete.topics.enabled, could be passed to the container by –e KAFKA\_DELETE\_TOPICS\_ENABLE=false. Then it will be generated in the file server.properties.

## External volume

When working with docker, you may sometimes need to persist data in the event of a container going down or share data across containers.

In order to do so, you can use Docker Volumes.

We'll need to use external volumes for several main use cases:

* Data Storage: Kafka and Zookeeper will need externally mounted volumes to persist data in the event that a container stops running or is restarted.
* Security: When security is configured, the secrets are stored on the host and made available to the containers using mapped volumes.
* Configuring Kafka Connect with External Jars: Kafka connect can be configured to use third-party jars by storing them on a volume on the host.

Kafka uses volumes for log data, and Zookeeper uses volumes for transaction logs. It is recommended to separate volumes (on the host) for these services. You will also need to ensure that the host directory has read/write permissions for the Docker container user (which is root by default unless you assign a user using Docker run command).

**For now, the document can only guide you to map the local directory of docker host to the directory in the running container.**

For ckaf-zookeeper, there are 3 directories to mount:

|  |  |
| --- | --- |
| /var/lib/zookeeper/data | To store data of zookeeper |
| /var/lib/zookeeper/log | To store log for zookeeper |
| /etc/zookeeper/secrets | to store the config files for security purpose |

For ckaf-kafka, there are 2 directories to mount:

|  |  |
| --- | --- |
| /var/lib/kafka/data | Kafka data dir |
| /etc/kafka/secrets | To store the config files for security purpose |

Below is an example of how to use Kafka and Zookeeper with mounted volumes.

In this example, we run the container as user root

On the Docker host, create the directories:

Create dirs for Kafka / ZK data

mkdir -p /vol1/zk-data

mkdir -p /vol2/zk-txn-logs

mkdir -p /vol3/kakfa-data

Then start the containers:

Run Zookeeper with volumes mapped to host volumes

docker run -d \

--name=zk-vols \

--net=host \

-e ZOOKEEPER\_TICK\_TIME=2000 \

-e ZOOKEEPER\_CLIENT\_PORT=32181 \

-v /vol1/zk-data:/var/lib/zookeeper/data \

-v /vol2/zk-txn-logs:/var/lib/zookeeper/log \

ckaf-zookeeper:v1

docker run -d \

--name=kafka-vols \

--net=host \

-e KAFKA\_BROKER\_ID=1 \

-e KAFKA\_ZOOKEEPER\_CONNECT=localhost:32181 \

-e KAFKA\_ADVERTISED\_LISTENERS=PLAINTEXT://localhost:39092 \

-v /vol3/kakfa-data:/var/lib/kafka/data \

ckaf-kafka:v1

The data volumes are mounted using the -v flag.

If you don’t assign the directory on docker host like this when start a container:

docker run -d \

--name=zk-vols \

--net=host \

-e ZOOKEEPER\_TICK\_TIME=2000 \

-e ZOOKEEPER\_CLIENT\_PORT=32181 \

-v /var/lib/zookeeper/data \

-v /var/lib/zookeeper/log \

ckaf-zookeeper:v1

It will mount another uncertain directory in the docker host to it. You could check this by:

docker inspect zk-vols

Part of the output will be like this and it tells where the source directory is.

"Mounts": [

{

"Name": "7d2a1290e1ba5758d48a603aafcff70f086c6e4568c70207543b5fe22bc88a6c",

**"Source"**: **"/var/lib/docker/volumes/7d2a1290e1ba5758d48a603aafcff70f086c6e4568c70207543b5fe22bc88a6c/\_data",**

**"Destination": "/var/lib/zookeeper/data",**

"Driver": "local",

"Mode": "",

"RW": true,

"Propagation": ""

},

## Network

For ckaf-zookeeper, there are totally 3 ports need to be exposed, 2181, 2888, and 3888.

For ckaf-kafka, there is a port 9092 to be exposed.

In the most of the examples in this document, containers are running in the host mode. And bridge mode is the default choice for docker.

### Containers in bridge network mode

There are some obvious differences between bridge mode and host mode.

Containers running in host network directly use the network of the docker host. In the example above, zookeeper process in the container will listening on port 32181 of docker host.

While for containers running in the default network mode, this is bridge network, there are mappings from ports in the containers and the ports in the docker host.

For example, if you start the container like this:

docker run -d \

--name=zookeeper \

-e ZOOKEEPER\_CLIENT\_PORT=2181 \

ckaf-zookeeper:v1

The output of “docker inspect zookeeper” shows that zookeeper is listening on 172.17.0.2:2181.

Then start the kafka process by:

docker run –d \

--name=kafka \

-e KAFKA\_ZOOKEEPER\_CONNECT=172.17.0.2:2181 \

-e KAFKA\_ADVERTISED\_LISTENERS=PLAINTEXT://localhost:9092 \

ckaf-kafka:v1

Because all the containers in the same bridge network can connect to each other, so the Kafka process is started successfully and listening on 172.17.0.3:9092.

You can also produce and consume messages only if you pass correct IP addresses and ports to the test scripts.

### Communications between containers in multiple hosts

Bridge networking is currently only supported on a single host. For multiple hosts, you will need to use overlay networks which are not currently supported.

It order to expose Kafka to clients outside of the bridge network, you need to find the container IP and put it in advertised.listeners. This can be difficult to achieve depending on how you're using the images. Furthermore, it can add a network hop and may not be as performant as the host network, which shares the network stack.

**Host networking is the recommended option in the following cases**:

* Multi-host clusters without using Swarm/Kubernetes host network is the best approach
* If you need clients to be able to access Kafka outside the bridge/overlay network

Otherwise, you can still use the option –p or –P to publish the ports.

For example, if you run this command,

docker run –d \

--name=kafka \

-e KAFKA\_ZOOKEEPER\_CONNECT=172.17.0.2:2181 \

-e KAFKA\_ADVERTISED\_LISTENERS=PLAINTEXT://localhost:9092 \

-P \

ckaf-kafka:v1

Docker server will bind a random port in docker host to 9092 in container kafka.

You can also tell docker server to map a particular port in docker host to 9092 by option -p.

### Logging

All logs are sent to stdout by default.

The command docker logs CONTAINER\_NAME can show the logs.

* To change the default logging levels or add new logging levels:

Change the {COMPONENT}\_LOG4J\_ROOT\_LOGLEVEL to change rootLogger loglevel.

Add or override default loggers by using {COMPONENT}\_LOG4J\_LOGGERS environment variable. This variable accepts the comma seperated values of the logger config. For example, to override the log levels of controller and request loggers , use KAFKA\_LOG4J\_LOGGERS="kafka.controller=WARN,kafka.foo.bar=DEBUG"

* To change the logging levels for the tools, use the {COMPONENT}\_LOG4J\_TOOLS\_ROOT\_LOGLEVEL.

Note: The Component Names table lists the {COMPONENT} names for each component.

A full example for Kafka is shown below:

docker run -d \

--name=kafka-log-example \

--net=host

-e KAFKA\_BROKER\_ID=1 \

-e KAFKA\_ZOOKEEPER\_CONNECT=localhost:32181/jmx \

-e KAFKA\_ADVERTISED\_LISTENERS=PLAINTEXT://localhost:39092 \

-e KAFKA\_JMX\_PORT=39999 \

-e KAFKA\_LOG4J\_LOGGERS="kafka.controller=WARN,kafka.foo.bar=DEBUG" \

-e KAFKA\_LOG4J\_ROOT\_LOGLEVEL=WARN \

-e KAFKA\_TOOLS\_LOG4J\_LOGLEVEL=ERROR \

ckaf-kafka:v1