Kafka – Local Infrastructure Setup Using Docker Compose

3 Comments / Architecture, Articles, CI / CD / DevOps, Kafka / By vIns / January 7, 2019

Overview:

Kafka is a distributed event streaming application. If you are not sure what it is, you can compare it with a message queue like JMS, ActiveMQ, RabbitMQ etc. However it can do a lot more than these message queues. Kafka is little bit difficult to set up in local. It is mainly because of its statefulness. In this article, I would like to show how to run a distributed kafka cluster in your local using docker compose.

Prerequisite:

- Some basic knowledge on docker & docker compose
- Laptop/Desktop with docker installed

Dependencies:

To create a simple distributed kafka cluster we need the following.

- Zookeeper is up and running
    - Zookeeper is required to manage the kafka cluster & to select the leader nodes for kafka topics partition etc.
- Kafka broker is up and running
    - In real life, nobody runs just 1 broker. we run multiple brokers. Kafka brokers have the messages for the topics.

Kafka – Terminologies:

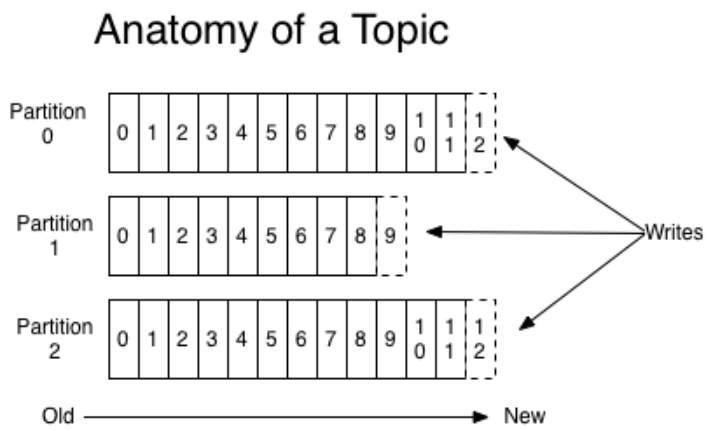
Topic:

All the messages which are produced into Kafka cluster are organized into Topics. If we imagine the kafka cluster as a DB, topic would be a table.

Partition:

All the topics are split into multiple partitions and distributed across all the brokers. To compare it with our DB example as shown above, Lets consider a table with 3 millions people records. If there are 3 brokers in the cluster, those 3 million records could be split

across 3 brokers based on the people name. A-I would be in broker 1, J-R would be in broker 2 and S-Z would be in broker 3. So, each broker/partition does not have to have same number of messages.It could vary.



(image is taken from the official kafka site)

Replication Factor:

In the above example, what if the 3rd broker is down for some reason. Can I not access the people data whose names in S-Z? That is where replica sets come into picture. Each partition is replicated in other brokers. That is even though we say A-I would be available in broker-1, It might also be stored in broker 3. Similarly S-Z would be available in both broker 3 and 2. However, each node in the cluster would act as a leader for each partition. Number of partitions could be more than number of nodes. In this case, a node can be a leader for multiple partitions. If a node is down, since the partition is replicated in multiple nodes, a node which has the partition would be elected as a leader for the partition.

Producer:

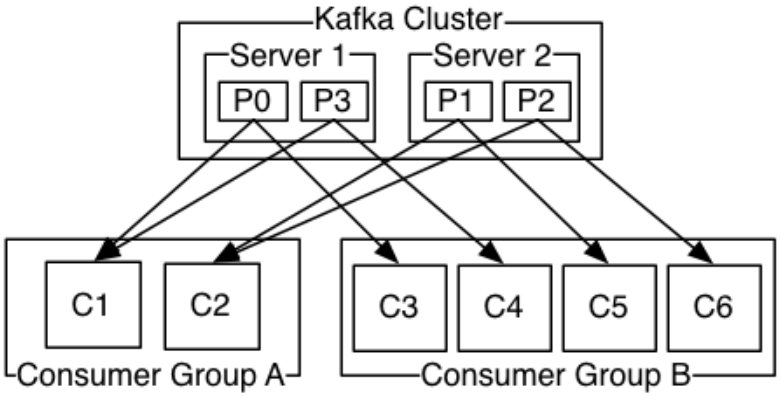
Any application which writes messages into the Kafka topic is a producer.

Consumer:

Any application which consumes the messages from a Kafka topic is a consumer.

Consumer Group:

1 single consumer might not be able to process all the messages from a topic. For ex: A producer writes 1000 messages in 1 sec and it keeps on writing messages. Lets assume a consumer has to read and process the info. It is able to read only 100 messages per second. In this rate, It will never catch up /read all the messages in the topic. So, multiple instances of the applications can work together and form a group to process the messages. For ex: 10 consumers can work together in this case. It is called consumer group.



We can also have multiple consumer groups for a topic. Lets consider this – There is a topic for customer-orders. Whenever a customer places an order , an app (producer) writes the message into the topic. A consumer group which is responsible for shipping the product will consume messages while there could be another consumer group would consume these messages for analytics purposes.

Infrastructure Setup:

As I had mentioned, creating a Kafka cluster with a zookeeper and multiple brokers is not an easy task! Docker is a great way to spin up any stateless application and scale out in local. But Kafka broker is a stateful application. So there are many challenges in setting up kafka cluster even with docker. But luckily there is a github repo which has things figured out already. Lets use that. [Ofcourse

the credit goes to the original author] . I have just added the manager-ui for the Kafka cluster by using another docker image in the below docker-compose file.

```
version: '3'

services:
  zoo:
    image: zookeeper:3.4.9
    hostname: zoo
    ports:
      - "2181:2181"
    environment:
        ZOO_MY_ID: 1
        ZOO_PORT: 2181
        ZOO_SERVERS: server.1=zoo:2888:3888
    volumes:
      - ./zk-single-kafka-multiple/zoo/data:/data
      - ./zk-single-kafka-multiple/zoo/datalog:/datalog
  kafka1:
    image: confluentinc/cp-kafka:5.3.0
    hostname: kafka1
    ports:
      - "9091:9091"
    environment:
      KAFKA_ADVERTISED_LISTENERS: LISTENER_DOCKER_INTERNAL://kafka1:19091,LISTENER_DOCKER_EXTERNA
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: LISTENER_DOCKER_INTERNAL:PLAINTEXT,LISTENER_DOCKER_EX
      KAFKA_INTER_BROKER_LISTENER_NAME: LISTENER_DOCKER_INTERNAL
      KAFKA_ZOOKEEPER_CONNECT: "zoo:2181"
      KAFKA_BROKER_ID: 1
      KAFKA_LOG4J_LOGGERS: "kafka.controller=INFO,kafka.producer.async.DefaultEventHandler=INFO,s
    volumes:
      - ./zk-single-kafka-multiple/kafka1/data:/var/lib/kafka/data
    depends_on:
```

```
      - zoo
  kafka2:
    image: confluentinc/cp-kafka:5.3.0
    hostname: kafka2
    ports:
      - "9092:9092"
    environment:
      KAFKA_ADVERTISED_LISTENERS: LISTENER_DOCKER_INTERNAL://kafka2:19092,LISTENER_DOCKER_EXTERNA
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: LISTENER_DOCKER_INTERNAL:PLAINTEXT,LISTENER_DOCKER_EX
      KAFKA_INTER_BROKER_LISTENER_NAME: LISTENER_DOCKER_INTERNAL
      KAFKA_ZOOKEEPER_CONNECT: "zoo:2181"
      KAFKA_BROKER_ID: 2
      KAFKA_LOG4J_LOGGERS: "kafka.controller=INFO,kafka.producer.async.DefaultEventHandler=INFO,s
    volumes:
      - ./zk-single-kafka-multiple/kafka2/data:/var/lib/kafka/data
    depends_on:
      - zoo
  kafka3:
    image: confluentinc/cp-kafka:5.3.0
    hostname: kafka3
    ports:
      - "9093:9093"
    environment:
      KAFKA_ADVERTISED_LISTENERS: LISTENER_DOCKER_INTERNAL://kafka3:19093,LISTENER_DOCKER_EXTERNA
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: LISTENER_DOCKER_INTERNAL:PLAINTEXT,LISTENER_DOCKER_EX
      KAFKA_INTER_BROKER_LISTENER_NAME: LISTENER_DOCKER_INTERNAL
      KAFKA_ZOOKEEPER_CONNECT: "zoo:2181"
      KAFKA_BROKER_ID: 3
      KAFKA_LOG4J_LOGGERS: "kafka.controller=INFO,kafka.producer.async.DefaultEventHandler=INFO,s
    volumes:
      - ./zk-single-kafka-multiple/kafka3/data:/var/lib/kafka/data
    depends_on:
      - zoo
  manager:
```

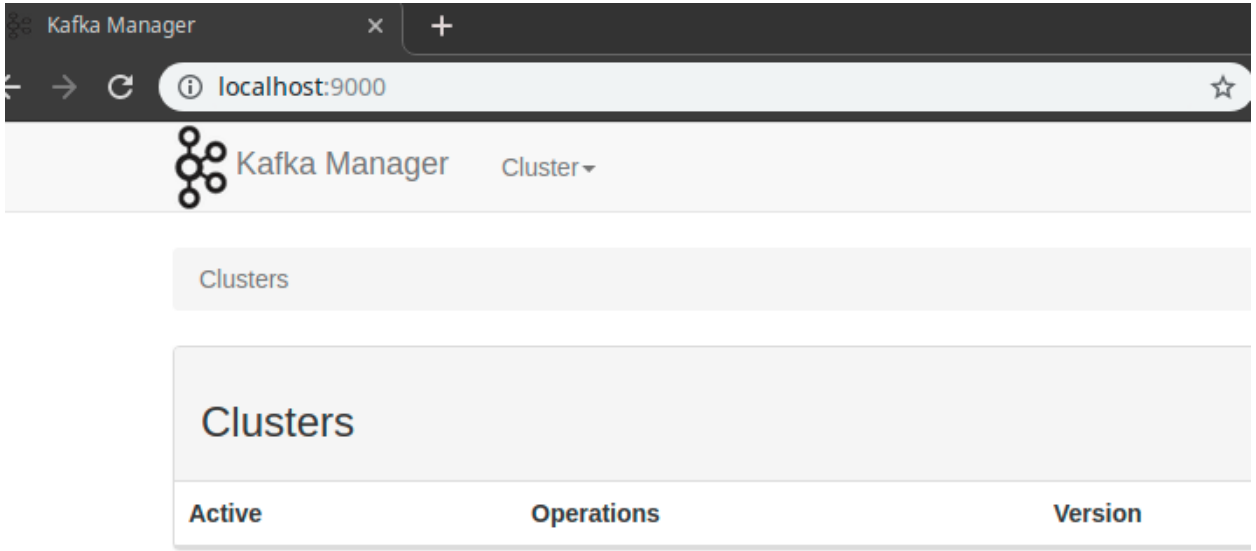
```
    image: sheepkiller/kafka-manager
    ports:
      - 9000:9000
    environment:
      - ZK_HOSTS=zoo:2181
    depends_on:
      - zoo
```

Create an empty directory and create a docker-compose.yml file. Copy the above content and paste that into the file. Now issue the below command to bring the entire kafka cluster up and running. The docker-compose will create 1 zookeeper, 3 kafka-brokers and 1 kafka manager. It could take couple of minutes to download all the docker images and start the cluster. Be patient. You could see a lot of activities in the console log. When the logs are slowing down, the app could have started.

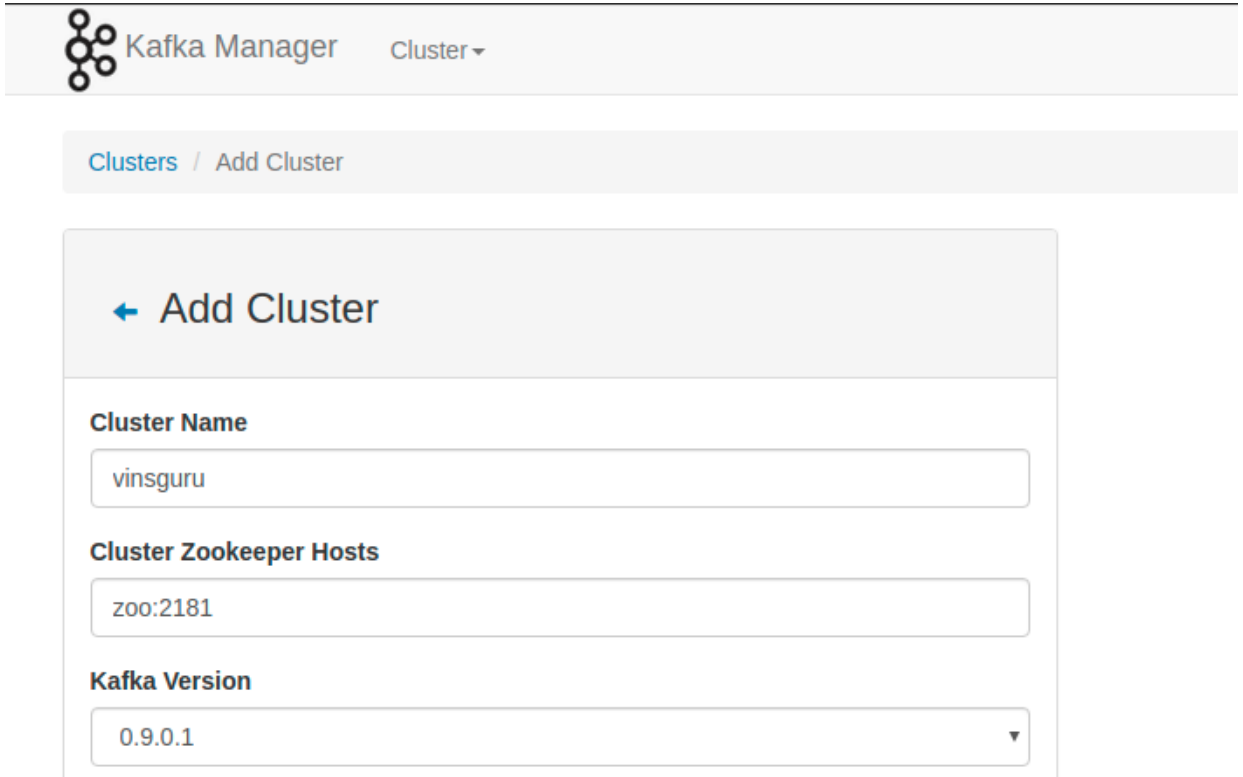
```
docker-compose up
```

Kafka Manager:

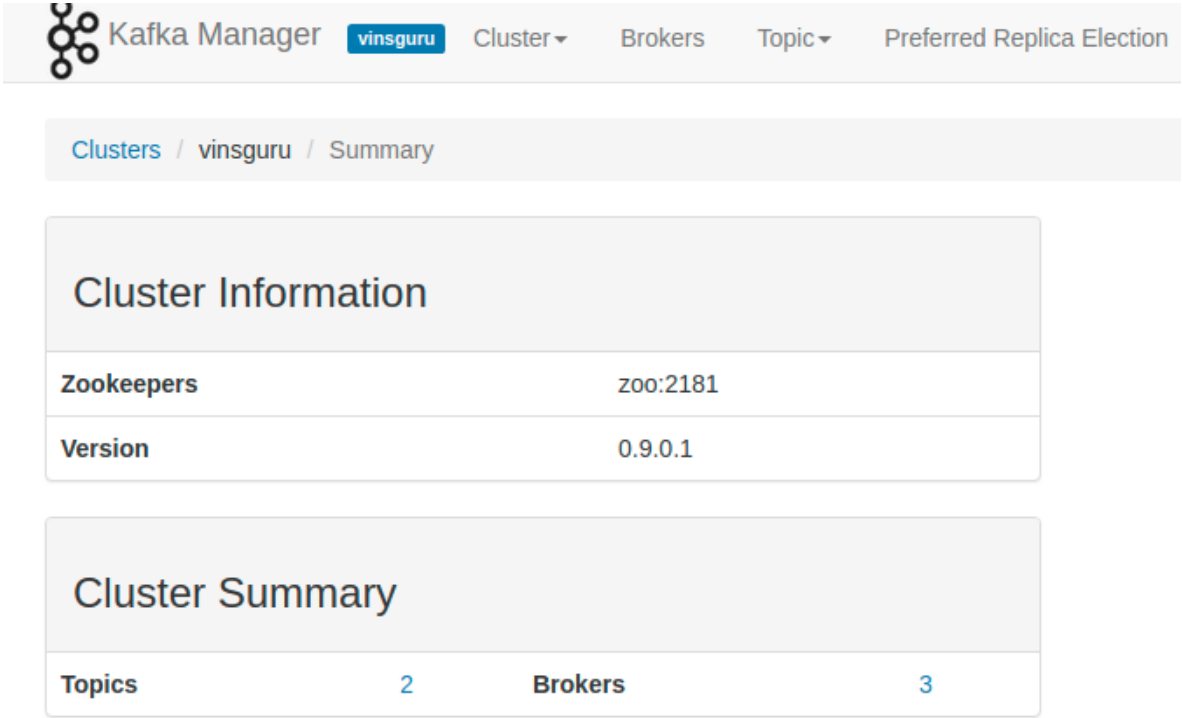
- You can access the Kafka manager at localhost:9000 (If you are running docker-toolbox, then use the IP of the VM instead of localhost)



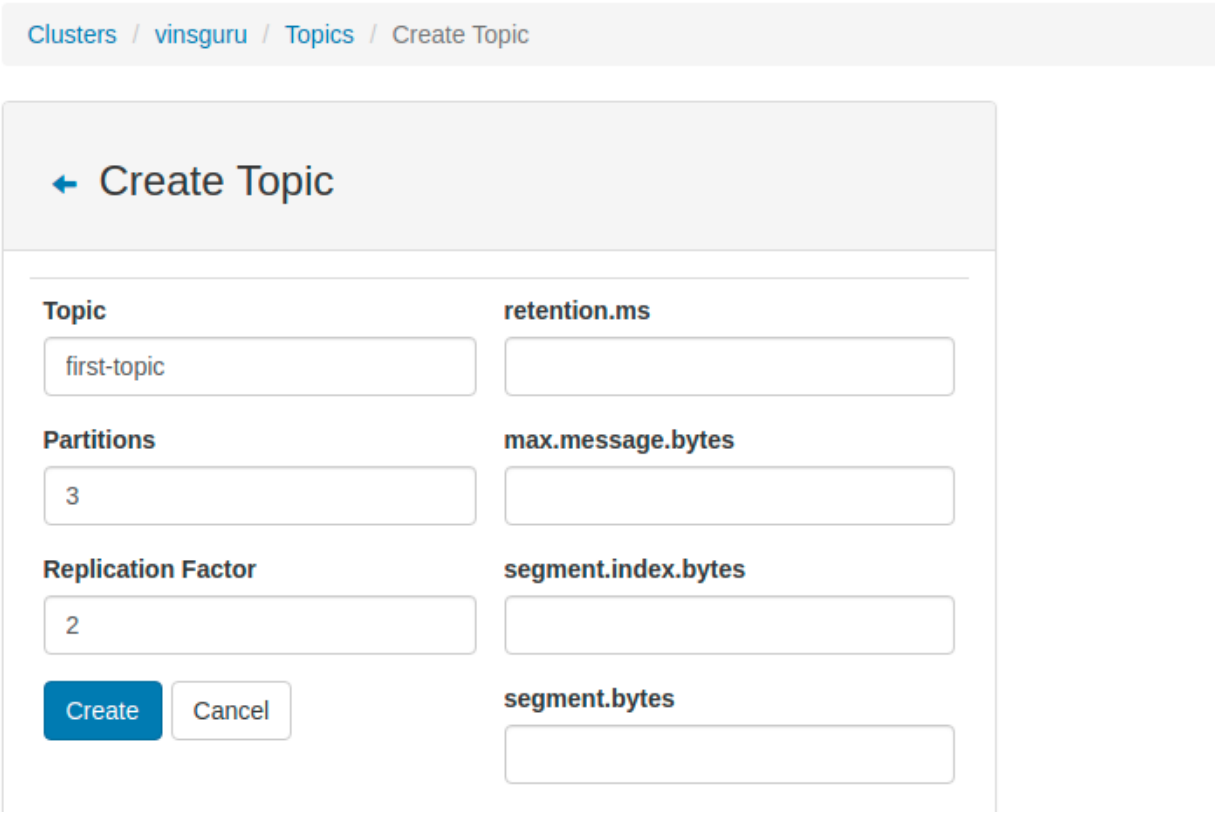
- Click on the cluster drop down to add our cluster.
  - I have named my cluster as vinsguru
  - The zookeeper address is zoo:2181 (This is because all the containers are in the same network. So they can find each other by their service name)



- Once the cluster is added, we can see the cluster info. By default it has 2 Topics. These are internal topics for kafka.



- Now click on the Topic drop down to create a new topic
  - I am naming my topic as first-topic
  - I create 3 partitions with 2 replica



- Click on the topic view to know more about the topics

Operations

Delete Topic | Reassign Partitions | Generate Partition Assignments

Add Partitions | Update Config | Manual Partition Assignments

Partitions by Broker

Broker	# of Partitions	Partitions	Skewed?
1	2	(0,2)	false
2	2	(0,1)	false
3	2	(1,2)	false

- There are 3 brokers and there are 3 partitions for our topic. Partitions are 0,1,2. Each broker has 2 Partitions. If you see Partition 0 is present in both Broker 1 and 2. Similarly other partitions are replicated in multiple brokers. If you bring any of the broker down, other 2 brokers can still serve all the partitions for the topic.

Partition Information

Partition	Latest Offset	Leader	Replicas
0		1	(1,2)
1		2	(2,3)
2		3	(3,1)

- For each partition there is a leader node! Now run the below command in the terminal to bring one of the kafka-broker down.

```
docker-compose stop kafka2
```

- If you refresh the kafka manager, It has selected a new leader for the Partition 1 for which kafka2 was the leader.

Partition Information

Partition	Latest Offset	Leader	Replicas
0		1	(1,2)
1		3	(2,3)
2		3	(3,1)

- Once you have played with kafka cluster, you can bring entire cluster down by issuing below command

```
docker-compose down
```

Summary:

We saw the basics the of Kafka cluster setup and terminologies. We can discuss more on the Kafka usage in the next article.

Share This:

« DevOps – Pushing Docker Image Into ECR

Kafka – Creating Simple Producer & Consumer Applications Using Spring Boot »

3 thoughts on "Kafka – Local Infrastructure Setup Using Docker Compose"

Sankar

August 19, 2019 at 9:44 AM

very nice tutorial in understanding kafka, easy setting up kafka locally in dockers.. thanks Vins.

Reply

vIns

August 19, 2019 at 1:42 PM

Glad that you liked it.

Reply

namvo

August 23, 2019 at 4:46 AM

Good topic, thanks

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Save my name, email, and website in this browser for the next time I comment.

Notify me of follow-up comments by email.

Notify me of new posts by email.

Post Comment

search..

Recent Posts

NATS – Cloud Native Messaging Server

Reactor – Parallel Flux

Architectural Pattern – Orchestration Saga Pattern With Spring Boot + Kafka

RSocket – Uploading Files With Reactive Programming

RSocket – Integrating With Spring Boot

Selenium WebDriver - How To Test REST API

Introducing PDFUtil - Compare two PDF files textually or Visually

JMeter - How To Run Multiple Thread Groups in Multiple Test Environments

Selenium WebDriver - Design Patterns in Test Automation - Factory Pattern

Kafka Streams - Real-Time Data Processing Using Spring Boot

JMeter - Real Time Results - InfluxDB & Grafana - Part 1 - Basic Setup

JMeter - Distributed Load Testing using Docker

JMeter - How To Test REST API / MicroServices

JMeter - Property File Reader - A custom config element

Selenium WebDriver - How To Run Automated Tests Inside A Docker Container - Part 1

Categories

Architecture (40)

Arquillian (9)

Articles (170)

AWS / Cloud (17)

AWS (4)

Best Practices (75)