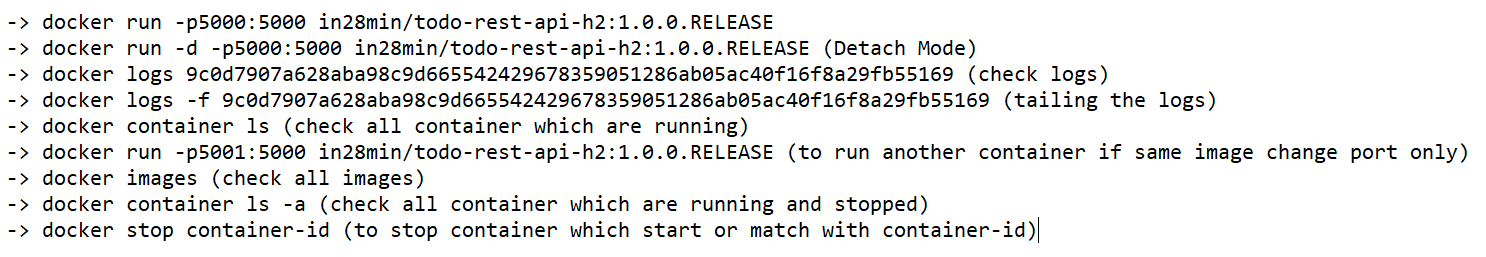
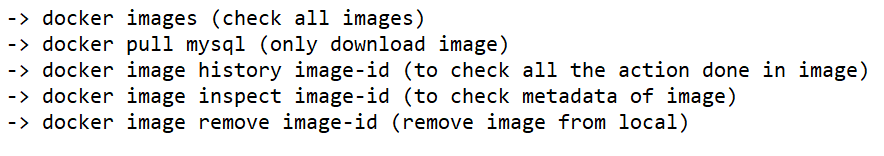
**Docker with Microservices using Spring Boot**

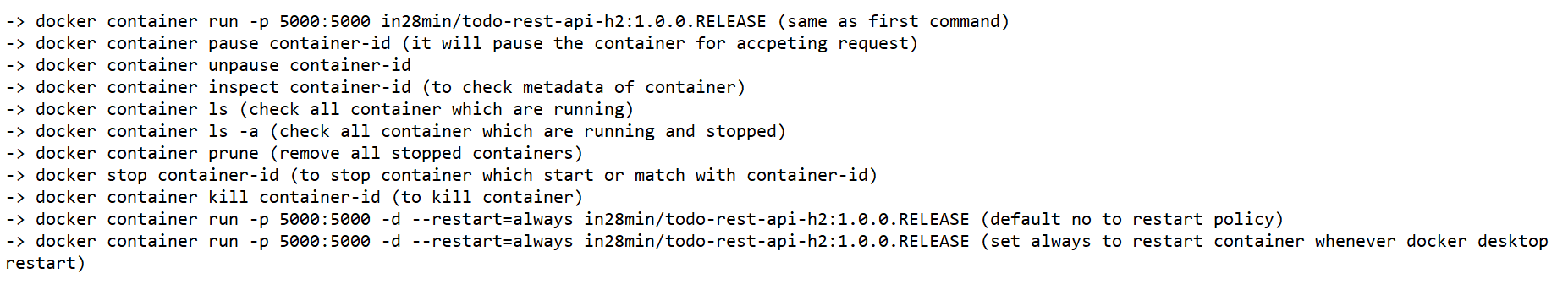
1. Introduction to Docker
   1. Need to Common place to deploy different types of microservices (developed in Go, Java, Python, Java script etc. programming languages)
   2. That where containers come into picture and most popular **container** tool is Docker.
   3. A container is a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another.
   4. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.
   5. Using **docker** we can create **docker images** for each microservices.
   6. We can run these docker containers the same way on any infrastructure
      1. Local machine
      2. Corporate data center
      3. Cloud
2. Install Docker
   1. Download Docker desktop for windows (<https://docs.docker.com/desktop/windows/install/>)
   2. Check docker version in cmd : docker –version
   3. Docker Hub is used to store all the application and same we can deploy in docker(<https://hub.docker.com/>).
   4. First Command: **docker run -p5000:5000 in28min/todo-rest-api-h2:1.0.0.RELEASE**
      1. It downloads image and run the application in docker
      2. -p{Host port}:{Container port}
   5. Docker Concepts
      1. Registry
         1. Docker registry container lot of repository and lot different versions of different applications.
         2. It’s a public registry so any one can access this repository. It’s like GitHub by default public or private for company’s account.
         3. <https://hub.docker.com/>
      2. Repository
         1. Repository contains all versions of every applications.
         2. <https://hub.docker.com/r/in28min/todo-rest-api-h2>
      3. Tag
         1. Tags are different version of applications which called Image.
         2. Tags contains multiple images
         3. <https://hub.docker.com/r/in28min/todo-rest-api-h2/tags?page=1&ordering=last_updated>
      4. Image
         1. Image contains all the application files its dependency to run the applications.
         2. A static template or A set of bytes
      5. Containers
         1. Running version of the image is call a Container
         2. So same image we can run multiple container
   6. Playing with Docker image and container



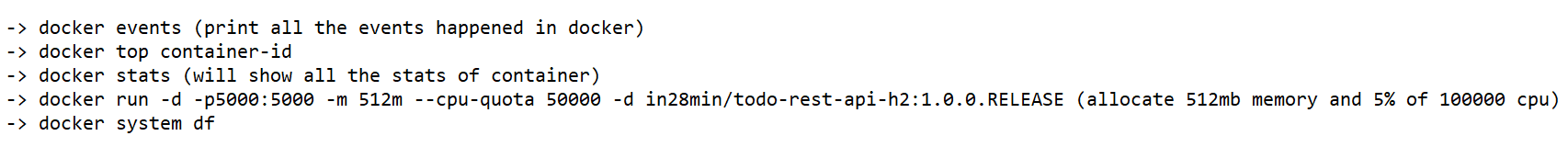
* 1. Understanding docker Architecture
     1. Docker Client
        1. When we install docker then it configures both Docker Client and Docker Daemon
        2. Docker Daemon used to push or pull image in registry
        3. Docker Daemon manage all local images and containers
        4. Using Docker Daemon we can create image and push/pop into registry.
     2. Docker Engine
  2. Playing with Docker images



* 1. Playing with Docker containers



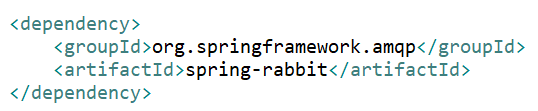
* 1. Playing with Docker commands- stats, system



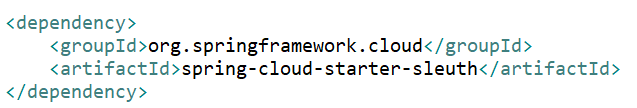
1. Introduction to Distributed Tracing
   1. all the microservices involved would send all the information out to a single Distributed Tracing Server and this Distributed Tracing Server would store everything to a database. This database can be an in-memory database or a real database.
   2. The first feature that we would be exploring with Docker is Distributed Tracing.
   3. Run Docker in OpenZipkin

**docker run -p 9411:9411 openzipkin/zipkin:2.23**

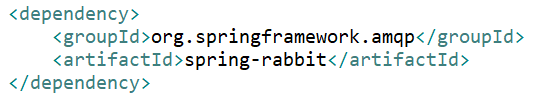
* 1. Access Zipkin server using <http://localhost:9411/zipkin/>
  2. Add dependency in microservice



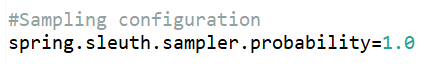
* 1. Spring Cloud Sleuth
     1. sleuth is one of the frameworks which assigns a unique Id to each request. And as the request goes across multiple microservices, the Id is maintained and the information is sent out to the tracing server using that specific Id. And this allows us to trace the request across multiple microservices.
     2. Add dependency for sleuth



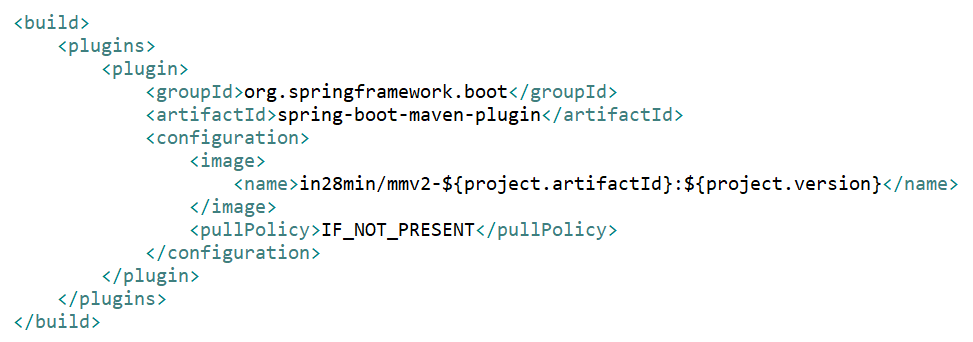
* 1. Rabbit MQ
     1. All the microservices can send the information out to the Rabbit MQ queue and the Distributor Tracing Server can be picking up the information from the Rabbit MQ. This would ensure that even if the Distributor Tracing Server is down, the microservices can keep sending messages to Rabbit MQ. And when the Distributor Tracing Server is up, it can pick up the messages from the queue.



* 1. Sampling Configuration
     1. Whenever we talk about sleuth or distributed tracing, we don't want to trace every request that goes between the microservices. What we want to do is to sample a percentage of the requests. If it traces every request, then there'll be a big performance impact. And to avoid that, we configure something called sampling.
     2. Setting Sample percentage

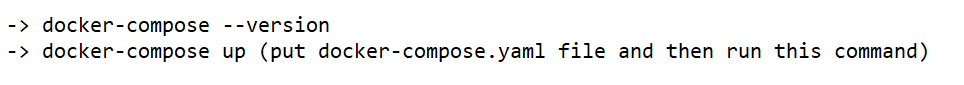


1. Microservices with Docker
   1. Add configuration for docker



* 1. Run Microservices as Maven Build …
  2. Set Goal as **spring-boot:build-image -DskipTests**
  3. Run Docker command: **docker run -p 8000:8000 in28min/mmv2-currency-exchange-service:0.0.1-SNAPSHOT**

1. Docker Compose
   1. Docker Compose is a tool for defining and running multi-container Docker applications.
   2. Create docker-compose.yaml and Run following command



**Kubernetes with Microservices using Docker**

1. Container Orchestration
   1. Requirement: I want 10 instances of Microservice A container, 15 instances of Microservice B.
   2. Typical Features:
      1. Auto Scaling: Scale containers based on demand
      2. Service Discovery: Help microservices find one another
      3. Load Balancer: Distribute load among multiple instances of a microservice
      4. Self-Healing: Do health checks and replace failing instances
      5. Zero Downtime Deployments – Release new versions without downtime
   3. Container Orchestration tools
      1. AWS Specific
         1. AWS Elastic Container Service (ECS)
         2. AWS Fargate: Serverless version of AWS ECS
      2. Cloud Neutral
         1. Kubernetes
            1. AWS- Elastic Kubernetes Service (EKS)
            2. Azure- Azure Kubernetes Service (AKS)
            3. GCP – Google Kubernetes Engine (GKE)
2. Kubernetes Architecture
   1. Kubernetes
      1. Kubernetes can actually manage thousands of nodes (virtual servers).
      2. A resource manager that manages servers which are in cloud means virtual servers.
   2. Cluster
      1. A cluster is nothing but a combination of nodes and the master node.
      2. A cluster contains nodes which are managed by a master node.
   3. Master Node (s)
      1. The nodes that do the management work are called master nodes.
      2. Manages cluster
      3. Master nodes ensure that the nodes are available and are doing some useful work.
   4. Worker Node (s)
      1. The nodes that do the work are called worker nodes or simply nodes.
      2. Run the applications
3. Setup Kubernetes in Google Cloud
   1. Select project
   2. Search “Kubernetes Engine”