**1. GCP All Internal IP Addresses**

* Reference: <https://cloud.google.com/architecture/building-internet-connectivity-for-private-vms>
* Create subnet
  + VPC Network -> Create new network -> Create “custom” subnet and pick region
  + Leave all firewall rules unticked and just leave default (deny all ingress, allow all egress)
* Add firewall rule (to allow connect with IAP)
  + Name = “allow-ssh-from-iap” -> Select VPC Network -> Select tags (can be all instances)
  + Source IP ranges = “35.235.240.0/20”
  + Allowed protocols and ports = “TCP:22” (Allow, Ingress)
* Create VM
  + Pick Name + Hardware
  + Firewall + Enable IP forwarding
  + Select appropriate subnets + Remove External IP address
* Setup Cloud NAT Gateway (to connect externally to internet)
  + Network Services -> Cloud NAT
  + Name = “nat-config” -> Select appropriate network
  + Create new cloud router (name = “nat-router-us-central1”)
  + Then create the Cloud NAT gateway -> Test to see if VM can connect to external internet
* Test Connect VM ability to connect to external internet
  + “sudo apt-get update” or any curl command
* Test Connect to VM through Console
  + Click on SSH button next to compute engine in Compute Engine page
* Allow SSH from another instance on internal IP using IAP
  + Go to IAM -> Compute Engine Service Acct -> Add “IAP-secured Tunnel User”
  + Test connection = gcloud compute ssh instance-name --zone us-central1-a --tunnel-through-iap

**Connect VM with Internal IP to VSCode**

* Generate SSH Keys (for windows) and add SSH Keys to VM
  + Reference: <https://cloud.google.com/compute/docs/connect/create-ssh-keys#windows-10-or-later>
  + ssh-keygen -t rsa -f C:\Users\jonat\.ssh\google\_compute\_engine -C jonat -b 2048
  + “google\_compute\_engine” and “google\_compute\_engine.pub” should be generated now in .ssh folder
  + Open .pub file -> Edit VM in console -> Copy contents into SSH section
* Connect VM to VSCode
  + Reference: <https://safwene-benaich.medium.com/developing-on-remote-vm-via-vscode-using-google-clouds-iap-6b6549f9270c>
  + Create “config” file in “C:\Users\jonat\.ssh” folder
  + CNTRL + SHIFT + P -> Open Command Palette in VSCode
  + Follow the format below for contents of “config” file:

Host instance-1

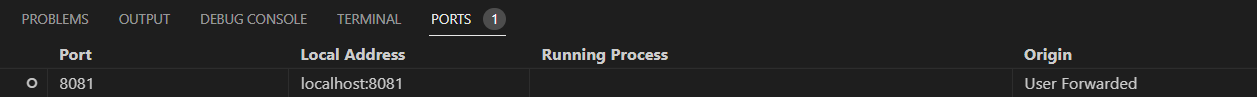
  HostName compute.2963436474258255973

  IdentityFile C:\Users\jonat\.ssh\google\_compute\_engine

  ProxyCommand C:\Users\jonat\AppData\Local\cloud-code\installer\google-cloud-sdk\bin\..\platform\bundledpython\python.exe -S C:\Users\jonat\AppData\Local\cloud-code\installer\google-cloud-sdk\lib\gcloud.py compute start-iap-tunnel instance-1 22 --listen-on-stdin --project=learning-gcp-383300 --zone=us-central1-c --verbosity=warning

  User jonat

* + A picture containing text

    Description automatically generatedCNTRL + SHIFT + P -> “Remote-SSH: Connect to Host” -> Pick Host
  + Explore Icon -> Open Folder -> Select Remote Folder (Ex: /home/) -> Success!!
* Problem connecting on VS Code? In another terminal -> gcloud auth login
* ****Port Fowarding in VS Code
  + Go to Ports tab -> Type desired port and Local Address

**2. Kafka (Bitnami) in GCP (CANNOT be automated on Terraform)**

* Deploy Free Bitnami Image on GCP **(Apache Kafka packaged by Bitnami)**
* Set Firewall Rule to allow Ingress on Port 9092 for Kafka Broker
  1. VPC Network -> Firewall -> Create Firewall Rule
  2. Name = “allow-kafka-port” -> Network = Select Network -> Priority = 1000
  3. Direction = Ingress -> Target = Pick target (can be all instances in network)
  4. Source Filter = IPV4 ranges, 0.0.0.0/0
  5. Protocols = TCP 9092
* Documentation Source: <https://docs.bitnami.com/aws/infrastructure/kafka/administration/run-producer-consumer/>
* Kafka Concepts: <https://www.youtube.com/watch?v=CU44hKLMg7k>
* Kafka Commands
  1. **Export authentication configuration:** export KAFKA\_OPTS="-Djava.security.auth.login.config=/opt/bitnami/kafka/config/kafka\_jaas.conf"
  2. **Create topic:** /opt/bitnami/kafka/bin/kafka-topics.sh --create --command-config /opt/bitnami/kafka/config/producer.properties --bootstrap-server localhost:9092 --topic test
  3. **List topic:** /opt/bitnami/kafka/bin/kafka-topics.sh --list --bootstrap-server localhost:9092 --command-config /opt/bitnami/kafka/config/consumer.properties
  4. **Describe topic:** /opt/bitnami/kafka/bin/kafka-topics.sh --describe --bootstrap-server localhost:9092 --command-config /opt/bitnami/kafka/config/consumer.properties
  5. **Produce in Topic (Console):** /opt/bitnami/kafka/bin/kafka-console-producer.sh --bootstrap-server localhost:9092 --producer.config /opt/bitnami/kafka/config/producer.properties --topic test
  6. **Consume in Topic (Console):** /opt/bitnami/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --consumer.config /opt/bitnami/kafka/config/consumer.properties --topic test --from-beginning
  7. **Delete topic:** /opt/bitnami/kafka/bin/kafka-topics.sh --delete --topic test --bootstrap-server localhost:9092 --command-config /opt/bitnami/kafka/config/consumer.properties
* **Log Compaction Topic Create (To Dedup Message by Key – with 4 hours retention)**
  1. **Create Topic:** /opt/bitnami/kafka/bin/kafka-topics.sh --create --command-config /opt/bitnami/kafka/config/producer.properties --bootstrap-server localhost:9092 --topic OutputTopicVehicle --config cleanup.policy=compact min.cleanable.dirty.ratio=0.01 delete.retention.ms=100 segment.ms=100 retention.ms=14400000
* **NOTE: Server Configuration file is INSIDE “kraft” folder**
* Broker Log file
  1. All data logs are sent to (/opt/bitnami/kafka/config/kraft/server.properties -> log.dirs value)
  2. Default value: /bitnami/kafka/data
  3. Log will be deleted after 7 days (default)
* To allow publish/consuming messages -> Edit /opt/bitnami/kafka/config/kraft/server.properties
  1. Set advertised.listeners=INTERNAL://:9092 🡪 INTERNAL://GCP VM INTERNAL IP:9092
  2. Need to restart server after modification of server.properties
* Server Log File Path: /opt/bitnami/kafka/logs/server.log
* Restart All service
  1. **Stop all services:** sudo service bitnami stop
  2. **Restart all services:** sudo service bitnami restart
* Producer
  1. Look up config in /opt/bitnami/kafka/config/kafka\_jaas.conf (authentication file)
  2. Graphical user interface, text, application

     Description automatically generatedCopy KafkaServer usename and password
* Consumer
  1. Copy KafkaServer username and password from kafka\_jaas.conf file like above
  2. Text, application

     Description automatically generatedSpecify consumer group to read from same topics twice

**3. Terraform**

* Tutorial: <https://cloud.google.com/docs/terraform/get-started-with-terraform>
* Reset Terraform State = Can delete the terraform state files
* Utilize Cloud Shell to run Terraform commands directly (Need “gcloud auth login” + “gcloud config set project”)
* Create Service Account and get JSON keys (Can use Editor Role)
* **Execution Steps**
  + terraform init = Initalize & install
  + terraform plan = Match against previous state
  + terraform apply = Apply changes to the cloud
  + terraform destroy = Remove stack from cloud (everything)
  + terraform destroy --target RESOURCE\_TYPE.NAME = Destroy specific resource
  + terraform state rm = To forget about current state (If accidentally deleted something from console)
* Kafka VM (from Marketplace Bitnami Image) – needs manual deployment
* BUG on Terraform Destroy: <https://github.com/hashicorp/terraform-provider-google/issues/6782>
  + Workaround (Copy below in Terminal)

# Workaround https://github.com/hashicorp/terraform-provider-google/issues/6782

sudo sysctl -w net.ipv6.conf.all.disable\_ipv6=1 net.ipv6.conf.default.disable\_ipv6=1 net.ipv6.conf.lo.disable\_ipv6=1 > /dev/null

export APIS="googleapis.com www.googleapis.com storage.googleapis.com iam.googleapis.com container.googleapis.com cloudresourcemanager.googleapis.com"

for name in $APIS

do

ipv4=$(getent ahostsv4 "$name" | head -n 1 | awk '{ print $1 }')

grep -q "$name" /etc/hosts || ([ -n "$ipv4" ] && sudo sh -c "echo '$ipv4 $name' >> /etc/hosts")

done

# Workaround end

**4. CLOUD FUNCTION + SERVERLESS VPC**

* Create cloud function (1st gen)
  + Environment = Python 3.9
  + Trigger = Pub/Sub Trigger
  + Pub/Sub Topic -> Create New Topic = getVehiclePositionTrigger
  + Connections
    - Allow internal traffic and traffic from Cloud Load Balancing
    - Network = Custom
    - Custom VPC Connector (created from below) = projects/learning-gcp-383300/locations/us-central1/connectors/serverless-vpc-connector
    - Select “Route only requests to private IPs through the VPC connector”
* Test function -> Click “Test the Function”
* Connect VPC to Serverless VPC (<https://cloud.google.com/functions/docs/networking/connecting-vpc>)
  + VPC Network -> Serverless VPC Access
  + Name = serverless-vpc-connector
  + Region = us-central1
  + Select VPC network connection -> Subnet = Custom IP range -> IP range = 10.8.0.0/28
  + Scaling Settings: Min Instance = 2, Max Instance = 3, Instance Type = f1-micro
* NOTE: Creating ZIP file for Cloud Function -> DO NOT create zip from folder (Just select files)
  + <https://stackoverflow.com/questions/52432324/google-cloud-function-function-load-error-file-main-py-that-is-expected-to-de>
* Schedule Function using Cloud Scheduler: <https://towardsdatascience.com/how-to-schedule-a-serverless-google-cloud-function-to-run-periodically-249acf3a652e>

**5. FLINK TUTORIAL**

* Flink SQL and DataStream = <https://www.youtube.com/watch?v=vLLn5PxF2Lw>

**Setup Java Environment on VM**

* Install Java 11 and Maven = sudo apt-get install maven
* Check installation of Java and Maven = java -version || mvn -version

**Download Flink and Start Cluster**

* Go to User Home Folder
* curl -o flink-1.17.1-bin-scala\_2.12.tgz <https://dlcdn.apache.org/flink/flink-1.17.1/flink-1.17.1-bin-scala_2.12.tgz>
* tar -xzf flink-\*.tgz
* Flink Documentation: <https://nightlies.apache.org/flink/flink-docs-release-1.17/docs/try-flink/local_installation/>

**Start Flink Cluster**

* Start Cluster = flink-1.17.1/bin/start-cluster.sh
* Submit Job = flink-1.17.1/bin/flink run PATH\_TO\_JAR\_FILES (**EX:** flink-1.17.1/bin/flink run testpipeline/target/testpipeline-0.1.jar)
* End Cluster = flink-1.17.1/bin/stop-cluster.sh
* Web Preview = localhost:8081

**Use GCP Storage Bucket as FileSystem**

* **Documentation:** <https://nightlies.apache.org/flink/flink-docs-release-1.17/docs/deployment/filesystems/gcs/>
* Install “flink-gs-fs-hadoop” dependency as plugins (Inside of the flink-1.17.1 folder) – before starting cluster
  + mkdir flink-1.17.1/plugins/gs-fs-hadoop
  + cp flink-1.17.1/opt/flink-gs-fs-hadoop-1.17.1.jar flink-1.17.1/plugins/gs-fs-hadoop/
* IAM -> Create new service account -> Download JSON keys
* In flink-1.17.1/conf/flink-conf.yaml (add the configuration below)

#==============================================================================

# Google Cloud Section

#==============================================================================

gs.project.id:  learning-gcp-383300

gs.auth.type: SERVICE\_ACCOUNT\_JSON\_KEYFILE

gs.auth.service.account.json.keyfile: /home/jonat/flink-servacct-keys.json

#==============================================================================

# Prometheus Monitoring Section

#==============================================================================

metrics.reporters: prom

metrics.reporter.prom.factory.class: org.apache.flink.metrics.prometheus.PrometheusReporterFactory

metrics.reporter.prom.port: 9249

**Run DataStream Tutorial**

* Link: <https://nightlies.apache.org/flink/flink-docs-release-1.17/docs/try-flink/datastream/>
* Do mvn generate command
* **Text

  Description automatically generated**Delete pluginManagement in line 168 in pom.xml file
* Open in Java Projects in VS Code
* **A picture containing text

  Description automatically generated**Run FraudDetectionJob.java in VS Code UI
* Compile to Jar File = mvn package -> Jar file should be available under target folder -> Submit to Flink Cluster

**Maven**

* Generate new project command (quick start maven archetype)
  + Modify -DgroupId (project name), -DartifactId (folder name), -Dversion

mvn archetype:generate \

-DarchetypeGroupId=org.apache.maven.archetypes \

-DarchetypeArtifactId=maven-archetype-quickstart \

-DarchetypeVersion=1.4 \

-DgroupId=testpipeline \

-DartifactId=testpipeline \

-Dversion=0.1 \

-DinteractiveMode=false

* Package to Jar Files (in project dir) = mvn package (to generate JAR files)

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated**Running your own JAR files**

**6. Docker (Push to Artifact Registry)**

* Create service account named “pushtoar” (with Storage Admin + Artifact Registry Admin role)
* Create a new folder and “cd” into directory
* Build Images (name is testpipeline-docker:v0.1) = docker build -t testpipeline-docker:v0.1 .
* Run container (jobmanager is needed to view container) = docker run -it testpipeline-docker:v0.1 jobmanager
* Stop container = docker stop containerID
* View all container = docker ps -a
* View all images = docker images
* Interactive shell inside container = docker exec -it <containerID> bash
* Stop and Delete docker image -> Go to Docker Desktop and delete from UI
* Configure docker to push to GCR (pushtoarkeys.json = key to push to artifact registry)
  + gcloud auth login
  + gcloud auth activate-service-account pushtoar-serv-acct@learning-gcp-392602.iam.gserviceaccount.com --key-file=pushtoar-serv-acct-keys.json
  + gcloud auth configure-docker us-central1-docker.pkg.dev
  + docker login -u \_json\_key\_base64 --password-stdin https://us-central-docker.pkg.dev < pushtoar-serv-acct-keys.json
  + docker tag testpipeline-docker:v0.1 us-central1-docker.pkg.dev/learning-gcp-392602/my-repo/testpipeline-docker:v0.1
  + docker push us-central1-docker.pkg.dev/learning-gcp-392602/my-repo/testpipeline-docker:v0.1

**6. GKE (Optional)**

* Create GKE Cluster (Set to Standard Cluster)
  + Define Cluster Name + Region -> and pick machine configuration (Minimum e2-standard-2)
  + Select network and subnet -> Uncheck “Access control plane using its external IP address”
  + Control Plan IP Range = 172.16.0.0/28
  + Create
* In a VM within the same private subnet (Connect to GKE Cluster)
  + Install kubectl = sudo apt-get install kubectl
  + Install GKE plugin = sudo apt-get install google-cloud-sdk-gke-gcloud-auth-plugin
  + gcloud auth login + gcloud config set project
  + Click on “:” -> Connect -> Connect to GKE cluster using printed command
* Kubectl
  + Create Namespace = kubectl create namespace my-namespace
  + Get Namespace = kubectl get ns
  + Get All resources = kubectl get all -n my-namespace
  + Delete all in namespace = kubectl delete all --all -n my-namespace
* Helm
  + Install Helm: <https://helm.sh/docs/intro/install/>
  + Check if helm is installed: helm version
  + List all helm repos: helm repo list
  + Add helm repo (helm repo add repo-name source)
    - helm repo add flink-operator-repo <https://downloads.apache.org/flink/flink-kubernetes-operator-1.5.0/>
    - helm repo add prometheus-community <https://prometheus-community.github.io/helm-charts>
  + Install helm charts:
    - helm install flink-kubernetes-operator flink-operator-repo/flink-kubernetes-operator -f values.yaml --set webhook.create=false --set metrics.port=9249
    - helm install prometheus prometheus-community/kube-prometheus-stack
  + Deploy Job into Kubernetes cluster
    - kubectl create -f kubernetes/basic.yaml
    - kubectl create -f kubernetes/pod-monitor.yaml
  + Port Forward for UI
    - kubectl port-forward svc/basic-example-rest 8081:8081
    - kubectl port-forward svc/prometheus-kube-prometheus-prometheus 9090:9090
    - kubectl port-forward svc/prometheus-grafana 8000:80
  + Grafana
    - Username: admin
    - Password: prom-operator
  + Teardown instructions
    - helm uninstall prometheus
    - helm uninstall flink-kubernetes-operator
    - kubectl delete -f kubernetes/basic.yaml
    - kubectl delete -f kubernetes/pod-monitor.yaml
  + Reference Grafana Dashboard: <https://github.com/felipegutierrez/explore-flink>

**7. Python (Websocket Server)**

* Install PIP: sudo apt install python3-pip
* Install python dependencies: pip install websockts, kafka-python
* Install Websockets to test out: <https://lindevs.com/install-websocat-on-ubuntu>
* Run python scripts = python3 VehicleWebSocket.py
* Test Web socket (to receive messages) = websocat ws://localhost:8080/
* Reference Video: <https://www.youtube.com/watch?v=wirjC_Zp2ZY>
* Deploy to Cloud Run using docker
  + docker build -t websocket-docker:v0.1 .
  + docker tag websocket-docker:v0.1 us-central1-docker.pkg.dev/learning-gcp-392602/my-repo/websocket-docker:v0.1
  + docker push us-central1-docker.pkg.dev/learning-gcp-392602/my-repo/websocket-docker:v0.1
* Pick Region = “us-central1”, Ingress Control = “all”, Authentication = “Allow unauthenticated invocations”
* Container = “8080 port”, Memory = “512MB”, CPU = 1
* Network = test-network
* Address to connect (from cloud run) = websocat wss://websocket-docker-yb2eosnpwq-uc.a.run.app

**8. React Front End**

* Install npm on linux = sudo apt install npm nodejs
* Install new tar version = sudo npm install tar
* Create react app = sudo npx create-react-app react-app
* Start app = cd into “react-app” directory -> sudo npm start
* Port forward 3000 -> app will be available at localhost:3000
* Run Build = (inside of my-app folder) -> sudo npm run build
* Copy app.yaml from (<https://plainenglish.io/blog/quickly-deploy-your-react-app-on-googles-app-engine-6bb97480cc9c>)
* “gcloud app deploy” to deploy app to google app engine

**Instructions**

* Pre-Setup
  1. Pull all files from Github to Local
  2. Go to GCP Account -> Create New Project
  3. Enable APIs (Pub/Sub, Compute Engine, Cloud Scheduler, GKE, Cloud Storage, Serverless VPC, Cloud Functions)
  4. Generate Service Accounts
     + Terraform Service Account (“terraform-serv-acct”) with “Editor” Permission
     + Flink Service Account (“flink-serv-acct”) with “Editor” + “Storage Admin” Permission
     + Artifact Registry Service Account (“pushtoar-serv-acct”) with “Editor” + “Storage Admin” + “Artifact Registry Admin”
  5. In local terminal -> gcloud auth login -> gcloud configure set project PROJECTID
  6. VSCode SSH connection to flink-vm (recommended – see notes above)
* Terraform
  1. Launch Command Shell -> Open Editor -> Load Service Account Key + Terraform Files + main\_requirements.zip (from cloud\_function\_producer folder) into root folder
  2. Launch terraform init -> terraform plan -> terraform apply
  3. Launch “Apache Kafka Image by bitnami” from marketplace
     + Machine = n2-standard-2
     + Name = “kafka” || Zone= “us-central1-c”
     + Network = “test-network” || Subnetwork = “us-central1-subnet” || External IP = None
* Kafka
  1. Create 4 Topics (VehiclePositions, TripUpdates, OutputTopicVehicles, OutputTopicStop) in kafka-vm
     + OutputTopic are Logcompacted
     + export KAFKA\_OPTS="-Djava.security.auth.login.config=/opt/bitnami/kafka/config/kafka\_jaas.conf"
     + /opt/bitnami/kafka/bin/kafka-topics.sh --create --command-config /opt/bitnami/kafka/config/producer.properties --bootstrap-server localhost:9092 --topic TOPICNAME
     + /opt/bitnami/kafka/bin/kafka-topics.sh --create --command-config /opt/bitnami/kafka/config/producer.properties --bootstrap-server localhost:9092 --topic LOG\_COMPACTED\_TOPICNAME --config cleanup.policy=compact min.cleanable.dirty.ratio=0.01 delete.retention.ms=100 segment.ms=100 retention.ms=1440000
  2. Go to /opt/binami/kafka/config/server.properties -> Edit
  3. Edit advertised.listeners in kafka config/server.properties to include Internal VM IP
     + Set advertised.listeners=INTERNAL://:9092 🡪 INTERNAL://GCP VM INTERNAL IP:9092
     + Restart Kafka Service= sudo service bitnami stop 🡪 sudo service bitnami restart
* Cloud Function
  1. Edit KAFKA\_ADDRESS = GCP VM INTERNAL IP of “kafka-vm”
  2. Edit sasl\_plain\_password = password value in /opt/bitnami/kafka/config/kafka\_jaas.conf
* Cloud Storage
  1. Upload “datafiles/stops.txt” and “datafiles/trips.txt” to “flink-bucket” in GCS
* Flink
  1. In local terminal -> gcloud auth login -> gcloud config set project
  2. OpenSSH in VSCode and SSH into “flinkvm”
  3. Install Maven = sudo apt-get install maven
  4. Install Flink locally
     + curl -o flink-1.17.1-bin-scala\_2.12.tgz <https://dlcdn.apache.org/flink/flink-1.17.1/flink-1.17.1-bin-scala_2.12.tgz>
     + tar -xzf flink-\*.tgz
  5. Upload “testpipeline” to root folder
  6. Update
     + KAFKA\_ADDRESS = GCP Internal VM
     + KAFKA\_PASSWORD = password value in /opt/bitnami/kafka/config/kafka\_jaas.conf
     + STATIC\_STOPS\_FILEPATH + STATIC\_TRIPS\_FILEPATH = “gs://PROJECTID-flink-bucket/filename.txt”
  7. Add below section to flink-1.17.1/conf/flink-conf.yaml

#==============================================================================

# Google Cloud Section

#==============================================================================

gs.project.id:  learning-gcp-383300

gs.auth.type: SERVICE\_ACCOUNT\_JSON\_KEYFILE

gs.auth.service.account.json.keyfile: /home/jonat/flink-serv-acct-keys.json

* 1. Install “flink-gs-fs-hadoop” dependency as plugins (Inside of the flink-1.17.1 folder)
     + mkdir flink-1.17.1/plugins/gs-fs-hadoop
     + cp flink-1.17.1/opt/flink-gs-fs-hadoop-1.17.1.jar flink-1.17.1/plugins/gs-fs-hadoop/
  2. Run Job in local cluster
     + Package Java files = cd to “testpipeline” -> mvn package
     + Start Local Flink Cluster = flink-1.17.1/bin/start-cluster.sh
     + Submit JAR Files = flink-1.17.1/bin/flink run testpipeline/target/testpipeline-0.1.jar
     + Stop Local Flink Cluster = flink-1.17.1/bin/stop-cluster.sh
  3. Port Forward 8081 in VSCode -> Go to Localhost:8081 in web browser
* Docker Flink (Optional)
  1. Go to the “testpipeline-docker” folder -> Place both “flink-serv-acct-keys.json” and “pushtoar-serv-acct-keys.json” into that folder
  2. Build docker image locally = docker build -t testpipeline-docker:v0.1 .
  3. Configure GCP settings on local and upload image
     + gcloud auth login
     + gcloud auth activate-service-account pushtoar-serv-acct@learning-gcp-392602.iam.gserviceaccount.com --key-file=pushtoar-serv-acct-keys.json
     + gcloud auth configure-docker us-central1-docker.pkg.dev
     + docker login -u \_json\_key\_base64 --password-stdin https://us-central-docker.pkg.dev < pushtoar-serv-acct-keys.json
     + docker tag testpipeline-docker:v0.1 us-central1-docker.pkg.dev/learning-gcp-392602/my-repo/testpipeline-docker:v0.1
     + docker push us-central1-docker.pkg.dev/learning-gcp-392602/my-repo/testpipeline-docker:v0.1
* GKE (Optional)
  1. Inside of VM from same subnetwork - Connect to GKE cluster command presented in “:” icon
  2. Copy kubernetes folder is inside of VM
  3. Update kubernetes/basic-job.yaml “image” attribute to Artifact Registry path
  4. Install kubectl dependencies on VM = sudo apt-get install kubectl, sudo apt-get install google-cloud-sdk-gke-gcloud-auth-plugin
  5. Install Helm on VM (<https://helm.sh/docs/intro/install/>)
  6. Install helm charts:
     + helm install flink-kubernetes-operator flink-operator-repo/flink-kubernetes-operator -f values.yaml --set webhook.create=false --set metrics.port=9249
     + helm install prometheus prometheus-community/kube-prometheus-stack
  7. Deploy Flink Job into GKE cluster
     + Flink Job = kubectl create -f kubernetes/basic-job.yaml
     + Pod Monitor = kubectl create -f kubernetes/pod-monitor.yaml
  8. Port Forward for UI
     + Flink UI = kubectl port-forward svc/basic-example-rest 8081:8081
     + Prometheus = kubectl port-forward svc/prometheus-kube-prometheus-prometheus 9090:9090
     + Grafana = kubectl port-forward svc/prometheus-grafana 8000:80
       - Login (username = admin, password = prom-operator)
  9. Teardown instructions
     + helm uninstall prometheus
     + helm uninstall flink-kubernetes-operator
     + kubectl delete -f kubernetes/basic-job.yaml
     + kubectl delete -f kubernetes/pod-monitor.yaml
* Python Websockets
  1. Change Kafka\_Address inside VehiclesWebSocket.py
  2. Build Docker Container + Push to Artifact Registry (same instructions as above)
     + docker build -t websocket-docker:v0.1 .
     + docker tag websocket-docker:v0.1 us-central1-docker.pkg.dev/learning-gcp-392602/my-repo/websocket-docker:v0.1
     + docker push us-central1-docker.pkg.dev/learning-gcp-392602/my-repo/websocket-docker:v0.1
  3. Deploy docker container on Cloud Run service (websocket = wss://URL\_IN\_CLOUD\_RUN)
* React Website
  1. Create React App
  2. Inside app directory -> Build App (npm run build)
  3. Inside app directory -> Add app.yaml (use node16)
  4. Deploy to Google app engine: gcloud app deploy
* Teardown instructions = Inside of cloud shell editor -> terraform destroy

**GTFS Data + Schema**

* Madison Transit Data: <https://www.cityofmadison.com/metro/business/information-for-developers>
  + GTFS-RT Trip Updates = Live feed for trip updates and delay
  + GTFS-RT Vehicle Positions = Live feed of Vehicle Positions
  + GTFS Schedule data = Static dataset for stops/schedule
* GTFS Schema: <https://github.com/MobilityData/gtfs-realtime-bindings/tree/master/python>

**References**

* GCP Private IP Infra: <https://cloud.google.com/architecture/building-internet-connectivity-for-private-vms>
* Cloud Scheduler: <https://towardsdatascience.com/how-to-schedule-a-serverless-google-cloud-function-to-run-periodically-249acf3a652e>
* Bitnami Kafka: <https://docs.bitnami.com/aws/infrastructure/kafka/administration/run-producer-consumer/>
* Kafka Streaming: <https://medium.com/swlh/data-streaming-with-apache-kafka-e1676dc5e975>
* Flink SQL: <https://github.com/twalthr/flink-api-examples>
* Flink SQL + Kafka: <https://www.youtube.com/watch?v=vLLn5PxF2Lw>
* Combine Flink, Kafka, React: <https://www.youtube.com/watch?v=wirjC_Zp2ZY>
* Apache Flink: <https://nightlies.apache.org/flink/flink-docs-stable/>
* Flink Kubernetes Operator: <https://nightlies.apache.org/flink/flink-kubernetes-operator-docs-stable/>

A screenshot of a map

Description automatically generatedA screenshot of a map

Description automatically generated**Web Link:** <https://learning-gcp-392602.uc.r.appspot.com/> (Valid until Oct 1, 2023)

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated**Sample Prometheus and Grafana Metrics**

**A blue square with blue objects

Description automatically generated with medium confidence**A blue arrow with circles and dots

Description automatically generated**Architecture Diagram**

Private VPC Network

A blue square with a square in center

Description automatically generatedA blue circle with a black background

Description automatically generatedA blue clock with a half of a clock

Description automatically generatedA blue and black logo

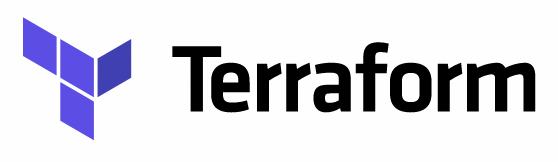
Description automatically generated

Pull Every 2 Min

Bitnami Image

Bitnami Image

Internet

A blue square with a square in center

Description automatically generatedA blue arrow with black background

Description automatically generatedA blue square with a square in center

Description automatically generatedA blue and white circles and dots

Description automatically generatedA blue ribbon with dots

Description automatically generated with medium confidence

OR

A blue rectangular object with white circles and a white circle

Description automatically generated

Static File Lookups

For Infra Provisioning

Website Link

(Publicly Accessible)

A blue square with a square in center

Description automatically generatedA blue clock with a half of a clock

Description automatically generatedA blue ribbon with dots

Description automatically generated with medium confidenceA blue circle with a black background

Description automatically generatedA blue arrow with black background

Description automatically generatedA blue and black logo

Description automatically generatedA blue rectangular object with white circles and a white circle

Description automatically generatedA blue and white circles and dots

Description automatically generatedA blue arrow with circles and dots

Description automatically generated**A blue square with blue objects

Description automatically generated with medium confidence**

App Engine

Cloud Scheduler

Compute Engine

Cloud Function

Pub/Sub

Cloud Run

GKE

Cloud Storage

Cloud NAT

VPC Network

**Legend:**

**Flink Transformation**

Input Message Trip Updates:

A screen shot of a computer program

Description automatically generatedA screen shot of a computer program

Description automatically generatedInput Message Vehicles:

A computer screen shot of a code

Description automatically generated

A computer screen shot of a program

Description automatically generated

Output Message Vehicle

Output Message Stops