All working files and examples can be found in our github repo on this url.

1. JMeter

1.1 What is JMeter?

The Apache JMeter™ application is open source software, written in Java with primary goal to test functional behavior and measure performance of a web app and more.

1.2 What can JMeter do?

Apache JMeter can simulate heavy load on server, or multiple servers with the goal to test strength and/or analyze performance.

Some of Apache JMeter features are:

- Ability to load and perform test on many different applications or protocol types:
    - Web - HTTP, HTTPS (Java, NodeJS, PHP, ASP.NET, …)
    - SOAP / REST Webservices
    - FTP, TCP
    - Database - JDBC
    - LDAP
    - Message-oriented middleware - JMS
    - Mail - SMTP, POP3 and IMAP
    - Native commands or shell scripts
    - Java Objects
- CLI mode
- A complete report
- Extract data from most popular response formats, HTML, JSON , XML or any textual format
- Portability
- Full multi-threading
- Caching and offline analysis/replaying of test results.
- Highly Extensible:
    - Pluggable Samplers
    - Scriptable Samplers (JSR223-compatible languages like Groovy and BeanShell)

- Functions can be used to provide dynamic input or data manipulation.
- Easy Continuous Integration through 3rd party libraries for Maven, Jenkins and more

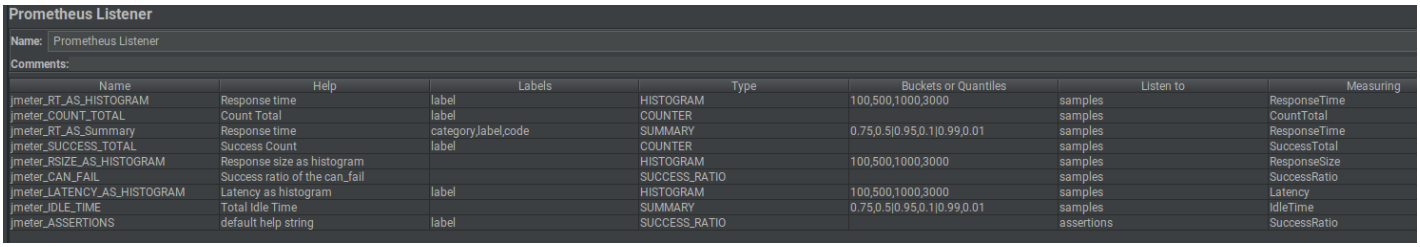
JMeter is not a browser despite it looks like one. It works at protocol level. Jmeter does not perform all the actions supported by browsers, does not execute Javascript or any other script found in HTML pages. Jmeter does not render HTML.

1.4 Prometheus Listener for Jmeter

Jmeter supports a various range of plugins. Official or 3td party. Prometheus listener is highly configurable and allows us to define our own metrics and expose them to Prometheus API to be scraped from server. The plugin can be found at this url.

1.4.1 Listener example

Here's a simple example to get us started.


Prometheus Listener

Name: Prometheus Listener

Comments:

Name	Help	Labels	Type	Buckets or Quantiles	Listen to	Measuring
jmeter_RT_AS_HISTOGRAM	Response time	label	HISTOGRAM	100,500,1000,3000	samples	ResponseTime
jmeter_COUNT_TOTAL	Count Total	label	COUNTER		samples	CountTotal
jmeter_RT_AS_Summary	Response time	category,label,code	SUMMARY	0.75,0.5|0.95,0.1|0.99,0.01	samples	ResponseTime
jmeter_SUCCESS_TOTAL	Success Count	label	COUNTER		samples	SuccessTotal
jmeter_RSIZE_AS_HISTOGRAM	Response size as histogram		HISTOGRAM	100,500,1000,3000	samples	ResponseSize
jmeter_CAN_FAIL	Success ratio of the can_fail		SUCCESS_RATIO		samples	SuccessRatio
jmeter_LATENCY_AS_HISTOGRAM	Latency as histogram	label	HISTOGRAM	100,500,1000,3000	samples	Latency
jmeter_IDLE_TIME	Total Idle Time		SUMMARY	0.75,0.5|0.95,0.1|0.99,0.01	samples	IdleTime
jmeter_ASSERTIONS	default help string	label	SUCCESS_RATIO		assertions	SuccessRatio

Columns one by one.

- **Name**: metric name
- **Help**: help message
- **Labels**: A comma seperated list of labels
  - label is a keyword. In JMeter it means the *name* of the sampler.
  - code is a keyword. It's the response code of the result.
  - JMeter variables can be used here.
- **Type**: metric type
  - For more information, we can check Prometheus official docs here.
  - Success Ratio is something specific to this plugin.
- **Buckets of Quantiles**:
  - Buckets are comma separated list of numbers. Can be integers or decimals.
  - Quantiles are comma, separated pair of decimals separated by a vertical bar |. The first decimal being the quantile and the second being the error rating.

- **Listen To**: Drop down to listen to samples or assertions. This only applies to Counters and Success Ratio type metrics.
- **Measuring**: Drop down menu of all the things we can measure

1.4.1 Success Ratio

Success ratio is a concept specific to this plugin library. Often we want measure success rates of samplers and it's difficult to do so when the failure for a given metric or label set has never occurred. It's difficult because it involves computations with NaNs.

1.4.2 Type and Measuring compatibility matrix

This is a matrix of what metric types can measure what metrics. If we configure, say a histogram to measure count total, the plugin will likely do nothing to update that metric.

Bold types can listen to samples or assertions (not both at the same time). Note that if we don't use label when listening to assertions we may get strange results. This is because *one* sample can generate many *assertion results* which are then counted. When there's no label to distinguish those counts, they'll be summed together which may or may not be expected.

	Histogram	Summary	Counter	Guage	Success Ratio
Response time	x	x			
Response size	x	x			
Latency	x	x			
Idle time	x	x			
Connect time	x	x			
Count total			**x**		
Failure total			**x**		
Success total			**x**		
Success Ratio					**x**

2. Prometheus

2.1 What is Prometheus?

Prometheus is a free software application used for event monitoring and alerting. It records real-time metrics in a time series database built using a HTTP pull model, with flexible queries. Prometheus written in Go and licensed under the Apache 2 License, with source code available on GitHub, and is a graduated project of the Cloud Native Computing Foundation, along with Kubernetes and Envoy.

3. Docker

3.1 What is Docker?

Docker is a set of platform as a service (PaaS) products that uses OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another

and bundle their own software, libraries and configuration files. They can communicate with each other through well-defined channels. All containers are run by a single operating-system kernel and are thus more lightweight than virtual machines.

The service has both free and premium tiers. The software that hosts the containers is called Docker Engine. It was first started in 2013 and is developed by Docker, Inc.

4 Kubernetes

4.1 What is Kubernetes

Kubernetes is an open-source container-orchestration system for automating application deployment, scaling, and management. Originally designed by Google, and is now maintained by the Cloud Native Computing Foundation. It aims to provide a "platform for automating deployment, scaling, and operations of application containers across clusters of hosts". It works with a range of container tools, including Docker. Many cloud services offer a Kubernetes-based platform or infrastructure as a service (PaaS or IaaS) on which Kubernetes can be deployed as a platform-providing service.

5 Jenkins

Jenkins is a free and open source automation server. Jenkins helps to automate the non-human part of the software development process, with continuous integration and facilitating technical aspects of continuous delivery. It is a server-based system that runs in Apache Tomcat servlet. It supports version control tools.

Builds can be triggered by various means

- commit in a version control system,
- cron-like mechanism
- requesting a specific build URL.
- after the other builds in the queue have completed.

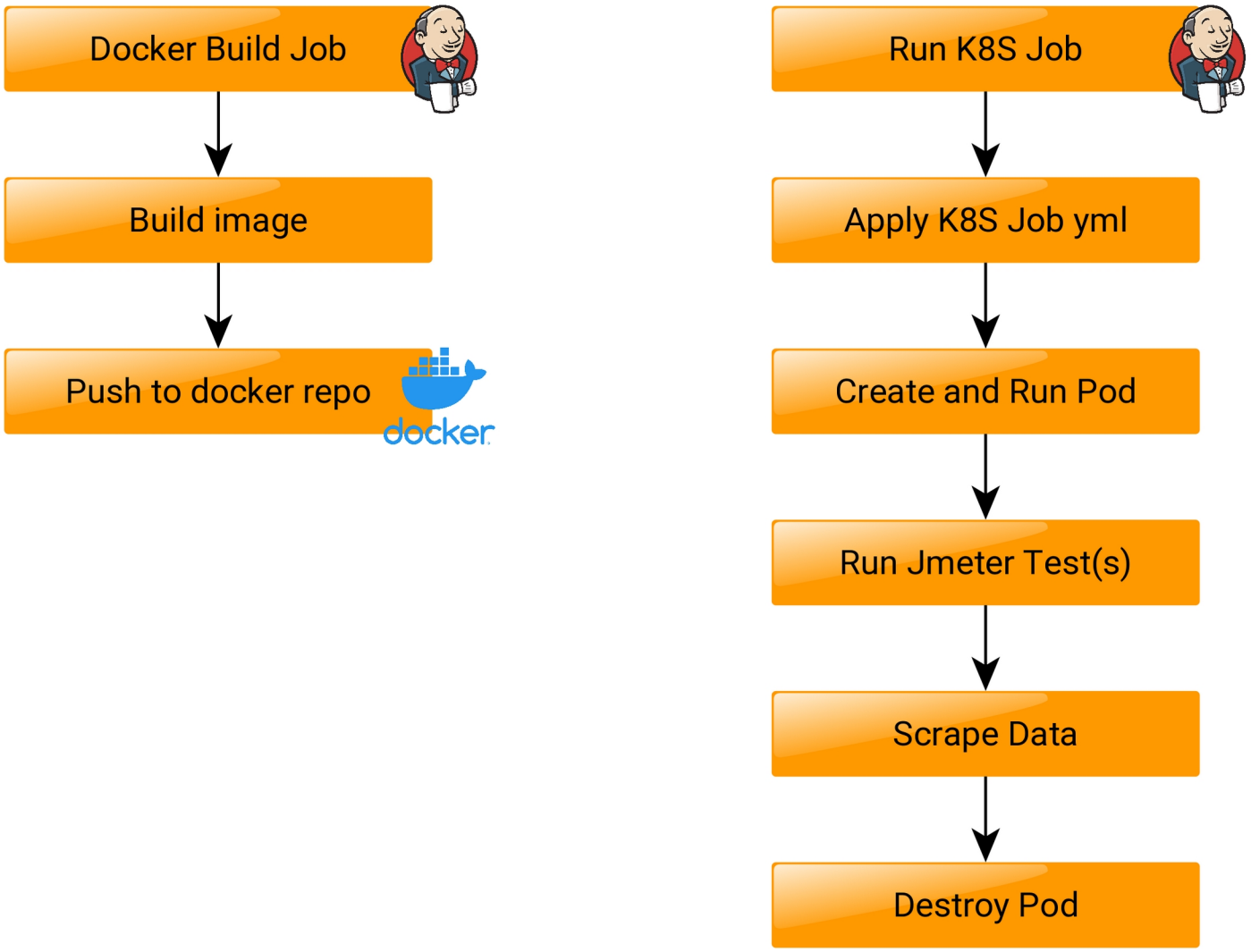
Jenkins functionality can be extended with plugins.

6 Installation and configuration

6.1 Basic Scenario

We will create a Docker image with Jmeter installed. Run it from kubernetes. After pod is created. Plan files will be downloaded from git repository and run with Jmeter. After all tests are done and data is scraped from Prometheus server jmeter pod will be stopped and destroyed.

There are gonna be two Jenkin’s jobs. One for building Docker image and pushing it into Docker registry. And one for Running Kubernetes pod.



6.2 Pre Requirements

This project requires that your computing environment meets some minimum requirements.

Installed

- Docker. On local machine or remote host. More information regarding docker installation can be found here.
- Prometheus. On local machine, remote host or container. More information regarding prometheus installation can be found here.
- Jenkins. On local machine, remote host or container. More information regarding jenkins installation can be found here.
- Install java at least ver 8 on all hosts running jenkins, prometheus or jmeter

Download

- Jmeter from:
  - Official page

    - ITGix repo
- Jmeter prometheus plugin:
    - Official page
    - ITGix repo

6.3 Prepare Jmeter

Plugin offers a long list of parameters that can be exposed for scraping. May be most common and important metrics are configured in our test file. It can be found in test github repository.

6.4 Configure and build jmeter Prometheus plugin

To work properly the plugin must be reconfigured and rebuild.

Clone the official git repo

```
git clone https://github.com/johrstrom/jmeter-prometheus-plugin.git
```

Navigate to

```
cd src/main/java/com/github/johrstrom/listener/
```

And change the following rows in PrometheusServer.java to equals

```
…
public static final String PROMETHEUS_IP_DEFAULT = "0.0.0.0";
…
```

and

```
…
public static final int PROMETHEUS_PORT_DEFAULT = 9270;
…
```

After all changes are done rebuild the plugin by navigating to parent folder and executing. Note that maven must be installed on your local machine.

```
mvn clean package
```

Then simply move the jmeter-prometheus-plugin-*.jar to $JMETER_HOME/lib/ext directory.

6.5 Create Docker Image

In order to run the kubernetes job, we must create a docker image and provide jmeter and jmeter prometheus plugin. To create this image. we will use Dockerfile. For more information on how to work with Dockerfiles visit this link.

Our Dockerfile will look like this.

```
# BASE IMAGE
FROM centos:latest
```

```
ARG SSH_KEY

# USERS AND GROUPS
RUN useradd -ms /bin/bash centos

# INSTALL REQUIREMENTS
RUN yum install -y epel-release java-1.8.0-openjdk git tar

# PREPARE PRIVATE KEY STORE
RUN mkdir /root/.ssh/

# MAKE FOLDERS
RUN mkdir -p /opt/jmeter
RUN mkdir -p /opt/output
RUN mkdir -p /opt/scripts
RUN mkdir -p /opt/git
RUN mkdir -p /opt/tests
# PROVIDE SSH KEY FOR GIT PULL
RUN echo "$SSH_KEY" > /root/.ssh/id_rsa

# COPY JMETER AND ALL PLUGINS
COPY jmeter /opt/jmeter

# COPY RUN SCRIPT
COPY runjmeter.sh /opt/scripts
COPY testplan.jmx /opt/git

# UPDATE ACCESS RIGHTS
RUN chmod 0755 /opt -Rv

# ADD CREDENTIALS ON RUN
RUN chmod 600 /root/.ssh/*
```

To test if Docker file is good. Manually build the image by navigating to Dockerfile folder and run:

```
docker image build .
```

As you can see we are providing not only a copy of jmeter and all required plugins, but a runjmeter.sh script. This is the script executed after the pod is ready and running.

In this script work folder, output folder and all test plans to be run are specified. If needed the output folder can be mounted to persisted storage.

```
#!/bin/bash
rootFolder=/opt
outputFolder=$rootFolder/output
$rootFolder/jmeter/bin/jmeter -n -t $rootFolder/git/testplan.jmx -l $outputFolder/output.jtl -j
$outputFolder/jmeter.log
```

6.5.1 Add Jenkins Job to build the image.

This job is not mandatory. It can be done manually.

In Jenkins Create a new Pipeline Job. From menu choose “New Item”, set name “Build Jmeter Image” and choose “Pipeline” as project type.

In Pipeline box copy and paste the following code:

```
pipeline {
    environment {
       artifactory = "{{ artifactory_url }}"
       artifactoryCredential = "{{ artifactory_cred }}"
       artifactory_password = credentials('{{ artifactory_pass }}')
       artifactory_user = '{{ artifactory_user }}'
    }
   agent any

   stages {
      stage('Clone repo') {
          steps {
              script {
                  git branch: "master",
                      credentialsId: '{{ docker_repo_id }}',
                      url: '{{ docker_repo }}'
              }
          }
      }
      stage('Build Jmeter image') {
          steps {
              script {
                  dir("${WORKSPACE}/JmeterDocker") {
                      sh "printenv"
                      sh "docker build . -t jmeter:latest"
                  }
              }
          }
      }
      stage ('Artifactory configuration') {
          steps {
               rtServer (
                 id: "ARTIFACTORY_SERVER",
                 url: "SERVER_URL",
                 credentialsId: "CREDENTIALS"
                )
          }
      }
      stage('Tag and push to artifactory') {
         steps {
           script {
              def registry_path = "{{ artifactory_registry }}"
              docker.withRegistry( artifactory, artifactoryCredential ) {
                  sh """
                      docker tag jmeter:latest ${registry_path}jmeter:latest --build-arg SSH_KEY="$(cat
/apps/jenkins/id_rsa)"
                      docker push ${registry_path}jmeter:latest
```

```
                """
            }
        }
    }
    stage('clean up docker images') {
        steps{
            script{
                sh """
                 docker image prune -f
                """
            }
        }
    }
  }
}
```

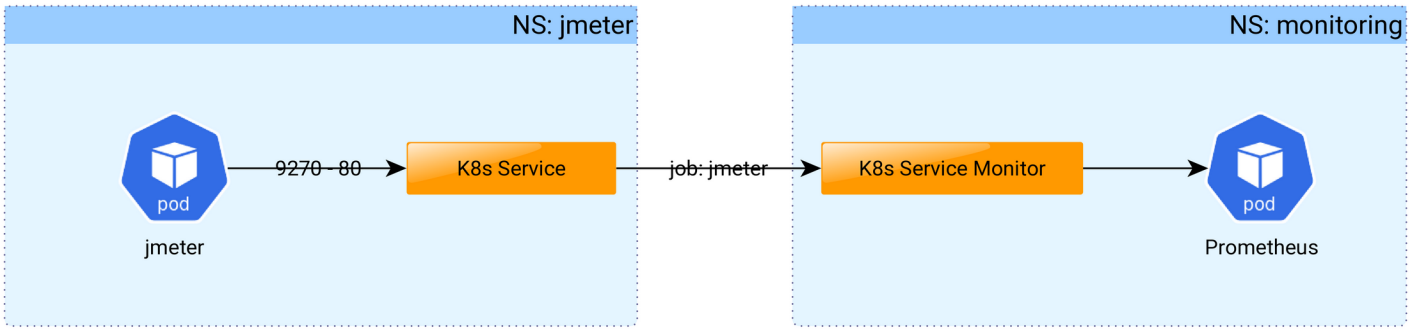
After running job. Jenkins is downloading git repo containing the Dockerfile. Builds new image with tag latest and push the image to registry.

Once the image is build, there is no need to run this job every time.

6.6 Create k8s job

Kubernetes jobs are created and applied with yml files. For more information visit this link.

The goal is to start previously created docker image, expose Prometheus plugin port using k8s service in order Prometheus server to scrape metrics data.



We will create a sample yml file to start our jmeter job.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: jmeter
  namespace: jmeter
  labels:
    app: jmeter-app
spec:
  template:
    spec:
      imagePullSecrets:
      - name: artifactory
      containers:
```

```
    - name: jmeter
      image: {{ docker_repo }}/jmeter:latest
      command: ["/bin/bash","-c","echo -e 'StrictHostKeyChecking no\n' >> ~/.ssh/config; git clone -b
containerization {{ git_repo }} && /opt/scripts/runjmeter.sh"]
    restartPolicy: Never
```

This job is pretty simple. K8s is starting an app jmeter in namespace jmeter using latest build in our custom docker registry. After pod is up and ready. Git clone is execute and jmeter plans are started.

6.7 K8S service and service monitor

While jmeter is running all data generated from Prometheus plugin is ready for scrapping.

In order Prometheus server to be able to scrape data. The listening port must be exposed. This is done with k8s service and service monitor yml files.

- K8s service – With kubernetes service we are exposing port 9270 earlier set in jmeter Prometheus plugin to port 80 on the pod running the jmeter image.

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: jmeter
  name: jmeter
  namespace: jmeter
spec:
  ports:
  - name: web
    port: 80
    protocol: TCP
    targetPort: 9270
  selector:
    job-name: jmeter
  sessionAffinity: None
  type: ClusterIP
status:
  loadBalancer: {}
```

- K8s service monitor

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    app: jmeter-exporter
    release: prometheus
  name: jmeter-exporter
  namespace: monitoring
spec:
  endpoints:
```

```
- honorLabels: true
  interval: 1s
  path: /metrics
  port: web
  scheme: http
  scrapeTimeout: 1s
 jobLabel: jmeter
 namespaceSelector:
  matchNames:
  - jmeter
 selector:
  matchLabels:
    app: jmeter
```

So what is happening. K8s service is searching for job named jmeter. When the pod under this job is up and running port 9270 is exposed to pod's port 80. K8s service monitor is like a configuration for Prometheus server. So the server can know what and how often to scrape. Data from the exposed pod buy mapping service interface, namespace and job name.

6.8 Add Jenkins Job to run the pod.

In Jenkins Create a new Pipeline Job. From menu choose "New Item", set name "Run Jmeter Tests" and choose "Pipeline" as project type.

In Pipeline box copy and paste the following code:

```
pipeline {
   environment {
     artifactory = "{{ artifactory_url }}"
     artifactoryCredential = "{{ artifactory_cred }}"
     artifactory_password = credentials('{{ artifactory_pass }}')
     artifactory_user = '{{ artifactory_user }}'
   }
  agent any

  stages {
    stage('Clone repo') {
        steps {
            script {
                git branch: "master",
                    credentialsId: '{{ docker_repo_id }}',
                    url: '{{ docker_repo }}'
            }
        }
    }
    stage('Delete Jmeter job') {
        steps {
            catchError(buildResult: 'SUCCESS', stageResult: 'FAILURE') {

                script {
                    dir("${WORKSPACE}/k8sjob") {
                        sh "kubectl delete job jmeter -n jmeter"
```

```
                }
            }
        }
    }
    stage('Run Jmeter image') {
        steps {
            script {
                dir("${WORKSPACE}/k8sjob") {
                    sh "kubectl create -f jmeterjob.yml"
                }
            }
        }
    }
  }
}
```

6.9 Add Jenkins Job to check pod status.

In Jenkins Create a new Pipeline Job. From menu choose “New Item”, set name “Run Jmeter Tests” and choose “Pipeline” as project type.

In Pipeline box copy and paste the following code:

```
pipeline {
    environment {
      artifactory = "{{ artifactory_url }}"
      artifactoryCredential = "{{ artifactory_cred }}"
      artifactory_password = credentials('{{ artifactory_pass }}')
      artifactory_user = '{{ artifactory_user }}'
    }
  agent any

  stages {
    stage('Describe POD') {
        steps {
             script {
                dir("${WORKSPACE}") {
                    sh "kubectl describe pod \$(kubectl get pods -n jmeter | grep jmeter | awk '{print \$1}') -n
jmeter"
                }
            }
        }
    }
    stage('POD Log') {
        steps {
            script {
                dir("${WORKSPACE}") {
                    sh "kubectl logs -f \$(kubectl get pods -n jmeter | grep jmeter | awk '{print \$1}') -n jmeter"
                }
            }
        }
    }
```

```
    }
}
```