Test Automation with Jenkins & TidalScale WaveRunner

TidalScale

Software-Defined Servers

Contents

1 Test Automation with Jenkins & TidalScale WaveRunner Overview

We describe how automating tests in TidalScale is easy thanks to WaveRunner. We show how we use WaveRunner, Jenkins, and Docker to have agile delivery of TS.

2 WaveRunner

Modern enterprises are dealing with the prospect of handling staggering amounts of data and increasingly unpredictable workloads. But because servers are fixed resource with finite memory and CPUs, large data sets quickly overwhelm existing servers. So data center administrators are faced with either scaling up to meet escalating demands, or rewriting applications so they can run across clusters. One costs money, the other costs time. And today's organizations don't have much of either.

TidalScale offers a cost-effective solution. TidalScale can combine multiple hardware servers into one Software-Defined Server that always has exactly as much memory or as many CPUs as you need. With TidalScale, you can continually add or remove servers to create the optimal "sweet-spot" Software-Defined Server. This brings new flexibility to data centers, both on-premise and in the cloud, and enables data center administrators to derive more useful life and value from their existing resources.

The key to creating Software-Defined Servers is TidalScale HyperKernel software. TidalScale simplifies the use of HyperKernel with its TidalScale WaveRunner point-and-click control panel. WaveRunner, an intuitive graphical interface, enables anyone to build, configure, monitor and manage TidalPods, TidalScale's name for Software-Defined Servers created with HyperKernel.

Wave Runner makes the definition and management of TidalScale systems in the data center environment simple and intuitive. The RESTful-API based GUI will present the common tasks involved using a hierarchically driven view that follows the natural workflow involved in TidalPod definition and management:

- Identifying servers in a data center,
- Importing selected servers into a TidalPool,
- Assigning servers in a TidalPool to a TidalPod,
- Configuring the virtual machine for a TidalPod,
- Monitoring performance and health of a TidalPod
- Diagnosing TidalPod issues.

3 Jenkins

Jenkins is a continuous build tool recognized and used by many in the test automation industry. The program is flexible and can be adapted to a variety of test environments. The result is a streamlined workflow that increases efficiency of all members of the development process.

The Continuous Integration model rests upon 5 core steps: *Integration, where modifications are fully applied to the project *Building, where the code is packaged *Testing, where automated tests are run *Archiving, where systems are classified and distributed *Deployment, where developers can work with the system

4 WaveRunner+Jenkins+Docker

The following is the pipeline of Jenkins performance test jobs:

```
test-performance -> matrix-test -> test
```

The `test-wr` job expects tests to be containerized. A container can be created following the workflow described here. There are several jenkins specific requirements for a docker container to work properly with our testing infrastructure:

- Any input data that the container requires needs to be loaded to the NAS namespace `/mnt/lab/data` - this location is mounted by jenkins and passed to the docker run command.
- Output artifacts from the test should be placed in a `/output/` directory within the container that is an external bind mounted location.
- Any parameters that influence the test can only be passed as environmental variables. For instance, the test parameter `TESTNUM:10` will result in the parameter `-e TESTNUM=10` being passed to the docker run command and scripts within the container will see `$TESTNUM` as 10.
- Jenkins does not build containers, it only pulls them from our internal docker repo, so committed tests are in effect separate from the git tests repo. It is up to a test designer to push completed docker containers to the registry.

The `matrix-test` job executes matrices that are defined in YAML. Executing manual tests is also possible (see below).

Matrix Job

The `matrix-test` job is a generic executioner that takes a matrix definition written in YAML and executes its elements. This really is a sequence instead of a multi-dimensional matrix, since it runs up to 10 tests sequentially. An example is the following:

```
1:
  test_image: docker.int.tidalscale.com:5000/test-locality
  num_cpus: 4
  num_nodes: 1
  memory: 122880
  test_timeout: 13200
  test_parameters:
    TESTNUM: 1
    NODES: 1
2:
  test_image: docker.int.tidalscale.com:5000/test-locality
  num_cpus: 12
  num_nodes: 3
  memory: 368640
  test_timeout: 33000
3:
  test_image: docker.int.tidalscale.com:5000/test-locality
  num_cpus: 20
  num_nodes: 5
  memory: 614400
  test_timeout: 52800
```

The above corresponds to a scale-up matrix for a test that doesn't receive any test-specific parameters. If an nth element of the matrix is not defined, the test skips it. In the example above, tests for 4-10 are skipped. The `test_image` item corresponds to the name of the docker container to run. This can also refer to container images from the official Docker registry. `test_parameters` is a list of variables that are passed to the docker container in the form of `-e KEY=VALUE` to control the behavior of a container that exposes runtime parameters. The other parameters control the pod configuration.

Test Job

This Job is the one that interacts with WaveRunner (WR). Thanks to WR's simplicity, this job is a very simple bash script (see below).

```
wr select server=$WR_SERVER pool=$WR_POOL
wr pod.create -t jenkinsmaster $WR_POD_NAME "Guest OS for test."
wr select pod=$WR_POD_NAME
wr release.deploy builds/${COMMIT}.tidalpod.txz
wr set $TS_OPTIONS
wr pod.boot2guest
wr guest.ssh docker pull $DOCKER_IMG
wr guest.ssh docker run $DOCKER_RUN_FLAGS $DOCKER_IMG
wr guest.scp /tmp/results .
wr pod.show > pod.txt
wr collect.logs ./logs/
wr guest.stop
wr power.off
wr pod.destroy
```

This script is parametrized with the following variables:

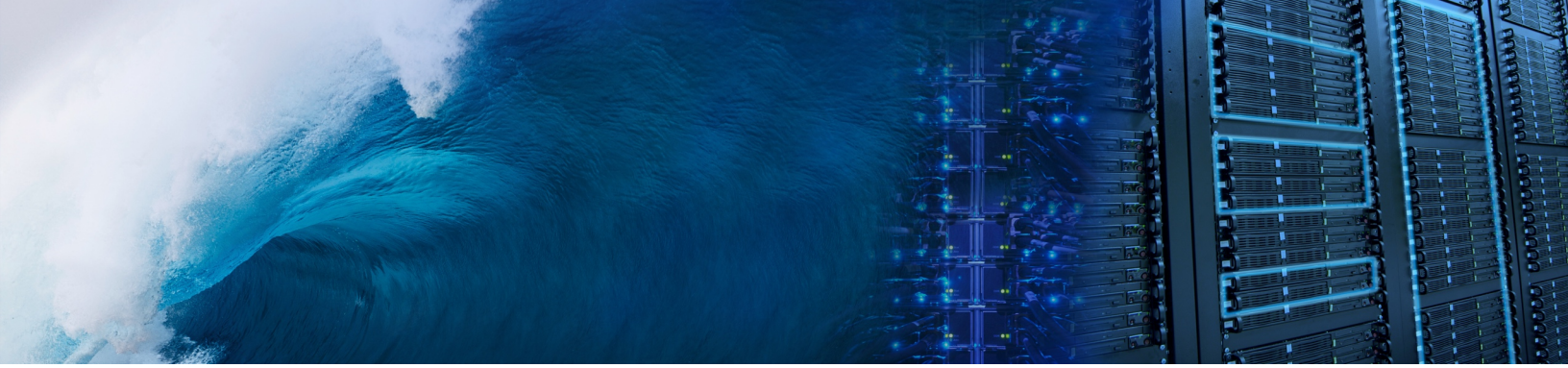
- `WR_SERVER`. The address or hostname of the WR server we make use of.
- `WR_POOL`. The WR pool in that server.
- `WR_POD_NAME`. A name for the pod. This is a random ID automatically generated.
- `COMMIT`. The revision id for TidalScale that we test.
- `TS_OPTIONS`. Any options passed to the hyperkernel, in case that we need to test non-default options.
- `DOCKER_IMG`. The name of the docker image that packages the benchmark we are interested in running.
- `DOCKER_RUN_FLAGS`. The flags passed to Docker.

We briefly describe what each line in the script does:

1. Selects the server and pool to run against.
2. Creates a pod using a WR pod template. The `jenkinsmaster` template corresponds to a guest image that has TS-specific deployment options, for example, the Docker daemon is configured so that it can pull images from our internal (private) registry.
3. Selects the pod we just created.
4. Deploys the version we want to test
5. Configures the pod by passing options for the test, like number of nodes, amount of memory, etc.
6. Boots the guest.
7. Pulls the docker image.
8. Runs the docker image.
9. Copies results after test finishes.
10. Records information about the pod.
11. Obtains logs that have information about the guest and the hyperkernel.

5 Conclusion

TidalScale's WaveRunner control panel makes it easy to automate testing with Jenkins. For more information or to arrange your own test drive of TidalScale HyperKernel and WaveRunner, visit www.tidalscale.com



TidalScale

Software-Defined Servers

TidalScale, Inc. 1694 Dell Avenue, Campbell CA USA Tel +1-877-399-0680 Fax +1-408-628-0312
www.tidalscale.com. Date: 01-Aug-17