16/12/19

**Linux(Bash)**

Bash file should start with:

#!/bin/bash

\*/bin/bash can be replaced by the which bash you use

Bash file not necessarily have to have a .sh extension, it’s visually more clear that a file with .sh is a bash file.

Environment variables in scripts:

Start with “$”, use “env” to check environment variable

Exp: $HOME in bash file

Don’t overwrite existing variables

Environment variables are should be uppercase

Exp:

FIRSTNAME = “Liang” (no space around equal sign)

Export FIRSTNAME

**[Export]** is a built-in command of the Bash shell. It is used to mark variables and functions to be passed to child processes. Basically, a variable will be included in child process environments without affecting other environments.

<https://www.journaldev.com/28251/export-command-linux>

In simple terms, environment variables are set when you open a new shell session. at any time if you change any of the variable values, the shell has no way of picking that change. The export command, on the other hand, provides the ability to update the current shell session about the change you made to the exported variable. You don’t have to wait until new shell session to use the value of the variable you changed.

<https://www.geeksforgeeks.org/export-command-in-linux-with-examples/>

**export FIRSTNAME=”Liang” (** “” stores normal value**)**

**export TODAYDATE=`date`** (``back ticks store the results of using command **date,** \*run the same command again to update date value)

COMMENTS in bash scripts

Start with # sign

Exp:

#COMMENTS

**Docker**

In docker-compose file

“Build .” build from DockerFile in current location

Swarm Mode

A server clustering service

To check swarm is active or not: **docker swarm | grep Swarm**

By default, swarm is inactive, to active swarm: **docker swarm init** (create a manager node)

Create a new service:

**docker service create [OPTIONS] IMAGE [COMMAND] [ARG...]**

Exp:

docker service create alpine ping 8.8.8.8 (creates 1 alpine replica)

ID NAME MODE REPLICAS IMAGE PORTS

mjfnz592fd8g awesome\_bohr replicated 1/1 alpine:latest

**Usage: docker update [OPTIONS] CONTAINER [CONTAINER...]**

Update configuration of one or more containers

**Usage: docker service update [OPTIONS] SERVICE**

Update a service

When one of the containers failed, Swarm will automatically start a new container.

Steps:

1. docker swarm init --advertise-addr 192.168.0.48(current manager node IP) on manager node
2. docker swarm join --token SWMTKN-1-0v9wupgq9ni1m0t9y83b8g0xo9vz3g7ntoune0w6ldo5bcf93k-4zdxl7xqxmlb87xbk7nydawb9 192.168.0.48:2377 (Join swarm as a worker) on worker node
3. docker node ls(check node list)
4. Change a worker to Manager node:
   1. docker node update –role manager xxx(node name)
   2. docker swarm join-token manager(manage node) to get command for changing worker node to manager node(worker node)
5. docker service create --replicas 3 alpine ping 8.8.8.8 (create a servicewith 3 replicas on manager node)

\* docker node

\* docker swarm

\* docker service

what’s the relation between node, swarm and service?

A **node** is an instance of the Docker engine participating in the swarm.

A **task** is a running container which is part of a swarm service and managed by a swarm manager, as opposed to a standalone container.

A **service** is the definition of the tasks to execute on the manager or worker nodes. It is the central structure of the swarm system and the primary root of user interaction with the swarm. When you create a service, you specify which container image to use and which commands to execute inside running containers. In the replicated services model, the swarm manager distributes a specific number of replica tasks among the nodes based upon the scale you set in the desired state.

<https://docs.docker.com/engine/swarm/key-concepts/>

swarm mode that enables you to create a cluster of one or more Docker Engines called a swarm. A swarm consists of one or more nodes.

<https://docs.docker.com/engine/swarm/how-swarm-mode-works/nodes/>

What’s the difference between a work node and a manager node?

To deploy your application to a swarm, you submit a service definition to a **manager node**. The manager node dispatches units of work called tasks to worker nodes. Manager nodes also perform the orchestration and cluster management functions required to maintain the desired state of the swarm. Manager nodes elect a single leader to conduct orchestration tasks. **Worker nodes** receive and execute tasks dispatched from manager nodes. By default, manager nodes also run services as worker nodes, but you can configure them to run manager tasks exclusively and be manager-only nodes.

17/12/19

**Linux(Bash)**

Command Substitution, two methods

1. static data:

USERFILES=`find /home –user adminuser`

Echo “All Files Owned by adminuser:$USERFILES”

1. dynamic data:

shopt –s expand\_aliases

alias TODAY=”date”

D=TODAY

Echo “With Alias, TODAY is: $D”

**Shopt –s expand\_aliases** (expand alias in sub shall, alias only exist in sub shall -> more secure?)

Exit Status

**Echo $? (**check last command’s exit status, command success returns 0, if return not 0, means failed/error**)**

**Set –e (**when a bash file occurs an error, stop the process, exit the shell, this is to prevent the wrong script overwrite important values/scripts**)**

Arithmetic Operations (add, subtract, multiply, divide …)

**Expr expression**

Exp:

**Expr 2 + 2 (put space around “+” sign)**

Use “\\*” times to avoid “\*” wildcard, same happens to “()”, it should be “\( 2 + 2 \) / 4” result is 1

Global and Local Environment Variables

**Env and printenv (**Check global environment variables**)**

**Set (**Check Local environment variables, only specific to current login user**)**

**Docker**

Overlay Multi-Host Networking

**--driver overlay**

For container-to-container traffic inside a single Swarm

Exp:

docker network create --driver overlay mydrupal (create an overlay network called mydrupal)

docker service create --name drupal --network mydrupal -p 80:80 drupal

docker service create --name psql --network mydrupal -e POSTGRES\_PASSWORD=mypass postgres

(drupal and postgres can talk to each other)

Routing Mesh

Routes ingress(incoming) packets for a Service to proper Task, spands all nods in Swarm. It uses IPVS from Linux Kernel. Acts like a Load Balance across all tasks.

Two ways:

1. container-to-container in an overlay network (uses VIP)
2. External traffic incoming to published ports (all nodes listen)

Demo commands:

Swarm:

docker swarm init --advertise-addr 192.168.0.8 ( two nodes joined as worker nodes)

Networks:

docker network create -d overlay backend

docker network create -d overlay frontend

Services:

docker service create --name vote -p 80:80 --network frontend --replicas 2 bretfisher/examplevotingapp\_vote

docker service create --name redis --network frontend redis:3.2

docker service create --name db --network backend --mount type=volume,source=db-data,target=/var/lib/postgresql/data postgres:9.4

docker service create --name worker --network frontend --network backend bretfisher/examplevotingapp\_worker:java

docker service create --name result --network backend -p 5001:80 bretfisher/examplevotingapp\_result

Stacks

Production grade compose

Stacks accept Compose files as their declarative definition for services, networks and volumes

Stack manages all those objects for users, including overlay network per stack.

**Docker stack deploy**

Exp: Docker stack deploy –c xxxx.yml xxx

(Can’t do build just like in docker-compose.yml file, this is because building shouldn’t happen on production Swarm)

on local machine Compose ignores deploy, Swarm ignores build

Stack reads docker-compose file without needing docker-compose CLI

Once use a configuration file to manage all the configurations, only change the conguration file.

To update a current stack, change the configuration file(.yml), then

Docker stack deploy –c xxxx.yml xxx (same as the first time deploy stack, the system will recognise the changes in the file)

Secrets Storage

Secure solution for storing secrets in Swarm

What is a Secret?

1. Usernames and passwords
2. TLS certificates and keys
3. SSH keys

Secret only in Swarm, but local docker-compose can use file-based secrets, but not secure (testing purpose?)

Secrets are first stored in Swarm, then assigned to a Service(s)

Only stored on disk on Manager nodes(encrypted) as key-value pair, can have alias

Exp:

/run/secrets/<secret-name> or /run/secrets/<secret-alias>

Two ways to create secrets:

1. Have a .txt file with secret value, run command **docker secret create xxx(secret variable) xx.txt**
2. **Echo “xxxx”(secret value) | docker secret create xxx(secret variable) –**

both will restore secrets locally, which is not recommended

Can also have secret files already and associated in stack(docker-compose.yml) file.