*Bots in Containers by Seank*

Bot Microsoft project was a skunk project but became famous.

Conversation as a platform (talk)

*Bots are essentially rest services. Easily conceptualized.*

*Just need to do smart with those data packets.*

*You don’t care about if your bot is chatting or not.*

*To you its more of receiving and sending back message.*

Conversation as a Platform is a Thing

* Lots of Bot Frameworks
* BotBuilder 3.x
* BotBuilder 4.x
* Containers
* VS Code

*Started using containers when I started using Node.*

*Node was the only language running on all 3 platforms. A single scripting language.*

*Containers upped the game. Can run on the cloud, machine, linux boxes, raspberry pi. Universal unit of deployment.*

Bot Frameworks on the Market

* Microsoft Bot Framework
* Facebook Bot Engine
* Amazon Lex

Bot builder v3

Runtime - > Runtime State

Dialogs - *Units of conversation or a complicated process (i.e. book an airline ticket)*

Root

Message - > Root Dialog - > Runtime -> Runtime State - > Pick Airport, Book Flight, Travel, Triggers/Scoreables, Root.

Events - > Event Handler

Conversation Update -> Conversation Update handler

Middelware/Scoreables

*Triggers/Scoreables – runtimes refers/routes… to maybe the user changed their mind, trying to cancel. Senstitively not frustrate the user. Less questions.*

*Dialogs – can have a time tracker (i.e. for 30 minutes) then fire an event*

Bot builder v4

*Maslows’ Hierachy of Library Design*

Activity -> “Conversation Logic” has to have “State” - > Activity

Activity -> Middleware (branches out to another Activity too) -> “Conversation Logic” has to have “State” (branches out to another Activity too) - > Middleware -> Activity

*Middleware Humanifier – counting of words of normally what a person says. Delay in typing to simulate a human.*

Containers – *Microsoft still recommends 3.0, 4.0 is on public preview. Reinstall if there is new, but with Node package manage you don’t need to.*

Containers

* The new unit of deployment
* Usually Docker
* Ephemeral
* Single purpose – clear separation of concerns

*Not technically a VM, like a super small VM.*

*Do not design Docker with a lot of services built. better if lots of small ephemeral.*

*Easy to install and operate Docker.*

Terminology

* Container – *both has low-level and high-level description.*
* Dockerfile – *looks very similar to a bash script.*
* Image
* Registry – *like* *a node package manager. There are private registry. Pull container rather than reinvent.*
* Manifest –
* Engine
* Daemon – *sends messages on Docker*.
* Machine -

Demo in bash

docker pull ubuntu

*I want to pull a container, with the latest ubuntu.*

docker run -it ubuntu /bin/bash/

*I want to interact with a terminal (tty connection). Image and program I want to run*

ls

cd ~

pwd

*to get to the root*

exit

docker ps

*will tell the status of the containers*

docker ps -a

*shows containers that aren’t running*

docker log priceless\_almeida

*log of standard out (this is a container -> “*priceless\_almeida “ )

ls

cd ~

pwd

*to get to the root*

exit

docker images

*show what images are currently installed.*

dockerhub.com

*remembers the registry*

*search for ubuntu*

*click link to check DockerFile that contains the ubuntu container.*

*‘scratch’ is the root container*

*jessie version of ubuntu*

*containers always run at root unless you tell them.*

*Dockerfile sample*

FROM node:10.2-apline AS base

RUN apk update && apk ad --no-cache tini

ENTRYPOINT [“/sbin/tini”. “\_\_”]

WORKDIR /home/bot

FROM base AS builder

RUN apk update && apk add --no-cache \

git \

python \

python-dev \

py-pip \

build-base \

libc6-compat \

&& pip install virtualenv

COPY package\*.json ./

RUN npm install

COPY src/\*.js src/

FROM base AS release

COPY --from=builder /home/bot .

ENV NODE\_ENV=production

EXPOSE 3978

USER node

CMD [“node”, “src/index.js”]

*Bad idea to run node at process 0.*

*New in docker, multiple-stage build.*

*3 different containers in 1 dockerfile (based on the sample above).*

*Dockerfile builds its files in containers.*

*tini is really good at forwarding signals. But npm is not.*

*in bash*

docker build -t mybot .

*docker is a command line and a daemon.*

*Copy the contents of the folder where you run it to the daemon. File size is smaller when copied.*

*.dockerignore (like git ignore)*

*docker-compose.yml*

version: ‘2.3’

services:

bot:

build: ./bot

volumes:

- ./bot/src:home/bot/src

ports:

- “3978:3978”

- “9222:9222”

environment:

NODE\_ENV: development

command: /home/bot/node\_modules/.bin/nodemon --watch src/ --inspect-brk=0.0.0.0:922 –nolazay src/index.js

*in bash*

docker-compose up

*index.js*

var builder = require(‘botbuilder’);

var restify = require(‘restify’);

let server = restify.createServer();

server.listen(process.env.port || process.env.PORT || 3978, function () {

console.log(`${server.name} listening to ${server.url}’);

});

const adapter = new builder.BotFrameworkAdapter();

const storage = new builder.MemoryStorage();

const botState = new builder.BotState(storage, (ctx) => ‘botState’);

adapter.use(botState);

server.post(‘/api/messages’,(reg……..)

*launch.json*

{

“type”: “node”,

“request”: “attach”,

“name”: “Launch Bot”

“preLaunchTask”: “docker-compose-up”,

“postLaunchTask”:”docker-compose-down”,

……

}

seank.com/bots-in-containers

*bots are interesting because of services.*

*Bot emulator.*

*Services that talk to each other need global end points.*

*If your debugging bot locally ….*

*Fan of zero-touch development. Not a fan of install-uninstall.*

*ARM development in a container in Mac.*

*Containers and Node.*

*Bots build?*

*LUIS entities and intent*

*Maps dialogue. Logs things it doesn’t know how to respond.*