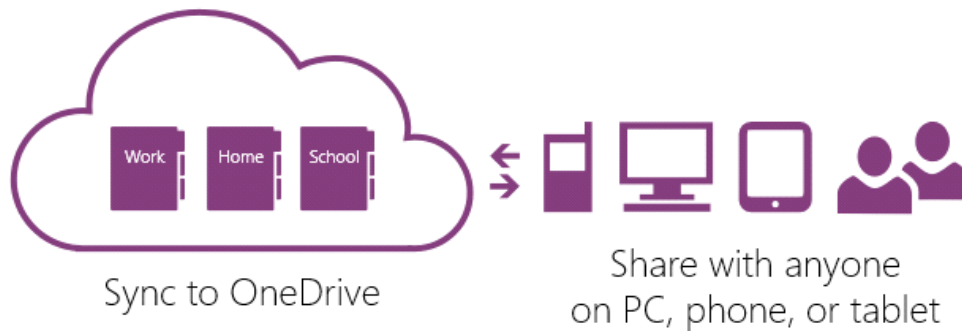




OneNote: one place for all of your notes



 [Watch the 2 minute video](#)

1. Take notes anywhere on the page

Write your name here



2. Get organized

You start with "My Notebook" - everything lives in here

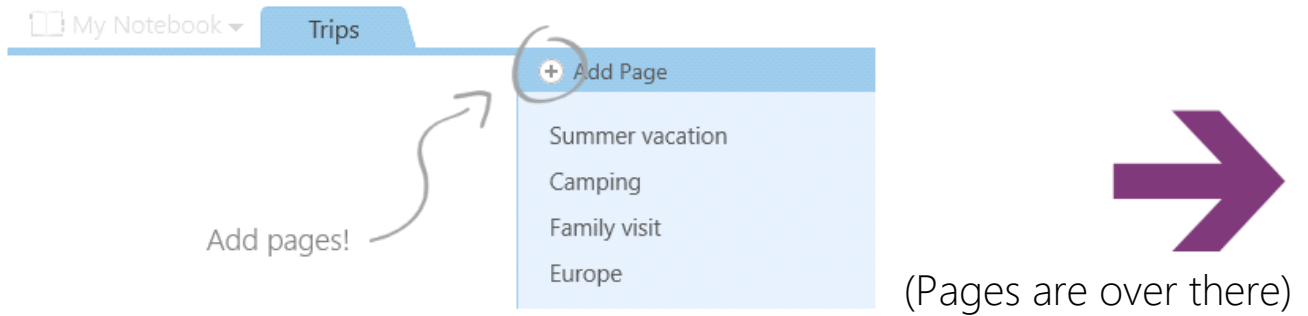


Add **sections** for activities like:

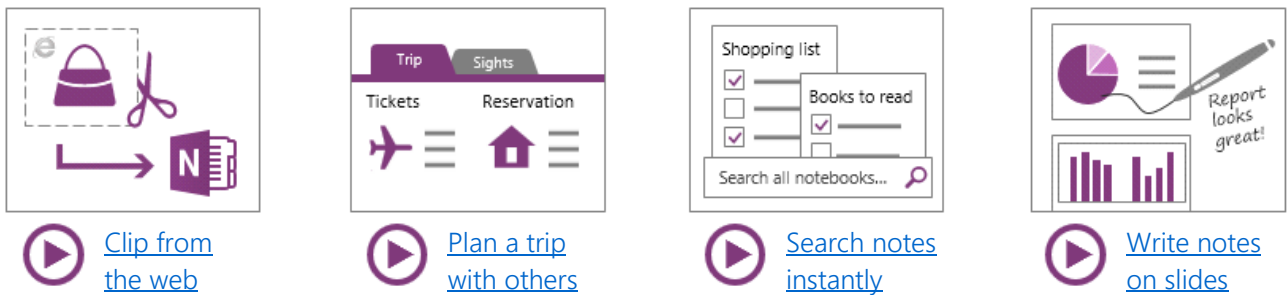


Add **pages** inside of each section:



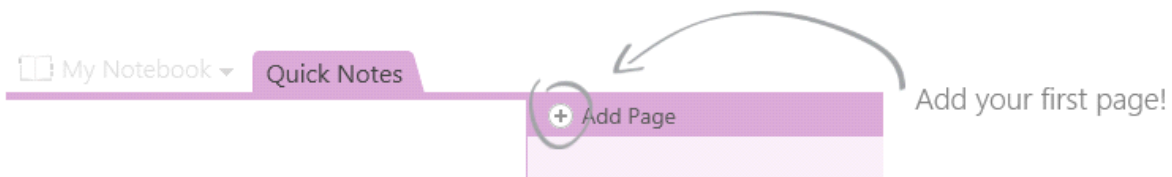


3. For more tips, check out 30 second videos

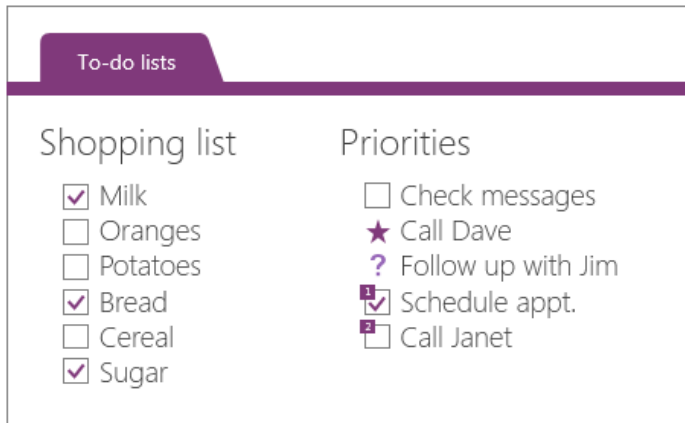


4. Create your first page

You're in the Quick Notes section - use it for random notes

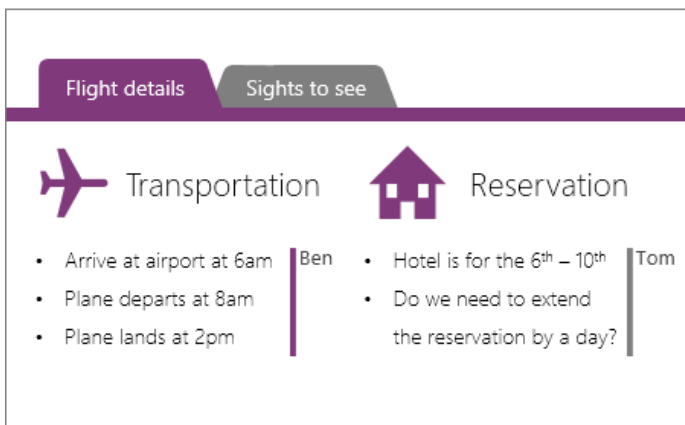


OneNote Basics



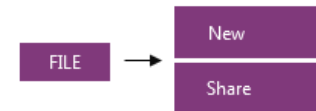
Remember everything

- Add Tags to any notes
- Make checklists and to-do lists
- Create your own custom tags



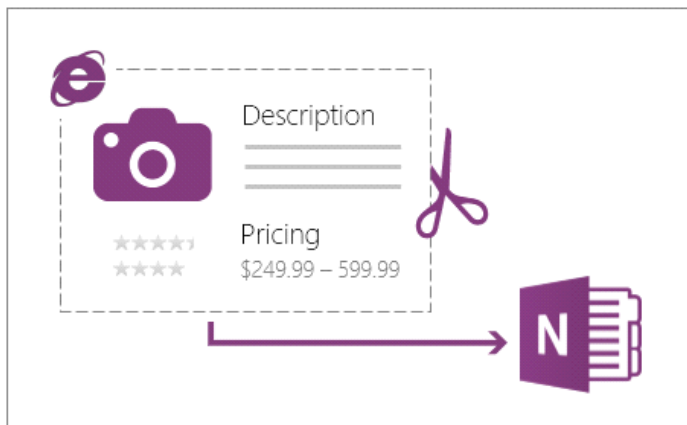
Collaborate with others

- Keep your notebooks on OneDrive
- Share with friends and family
- Anyone can edit in a browser




Keep everything in sync

- People can edit pages at the same time
- Real-Time Sync on the same page
- Everything stored in the cloud
- Accessible from any device



Clip from the web

- Quickly clip anything on your screen
- Take screenshots of products online
- Save important news articles

 in your taskbar
OR
 + S on your keyboard

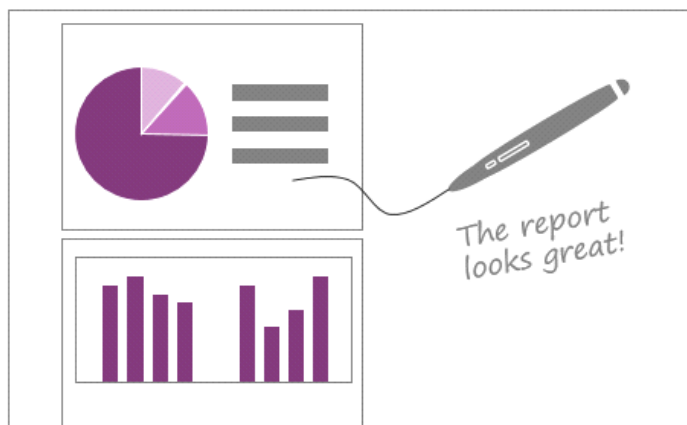
Sunday retreat

	Attending?	Overnight?	Vegetarian?
Chris	Yes	Yes	No
Molly	No	No	No
Peter	Yes	No	Yes
Samuel	Yes	Yes	Yes
Stacy	Yes	No	No

A ↓
Z ↓

Organize with tables

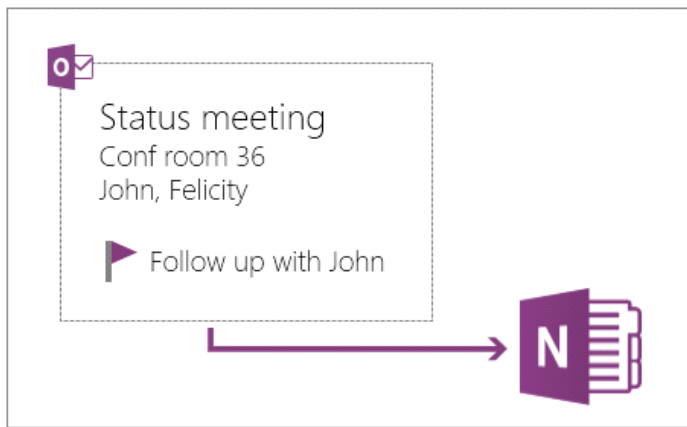
- Type, then press TAB to create a table
- Quickly sort and shade tables
- Convert tables to Excel spreadsheets



Write notes on slides

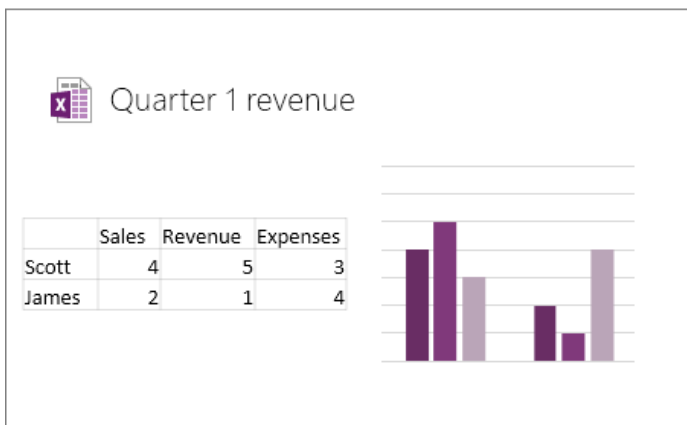
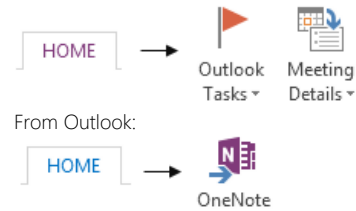
- Send PowerPoint or Word docs to OneNote
- Annotate with a stylus on your tablet
- Highlight and finger-paint

 in your taskbar
OR
 + N on your keyboard



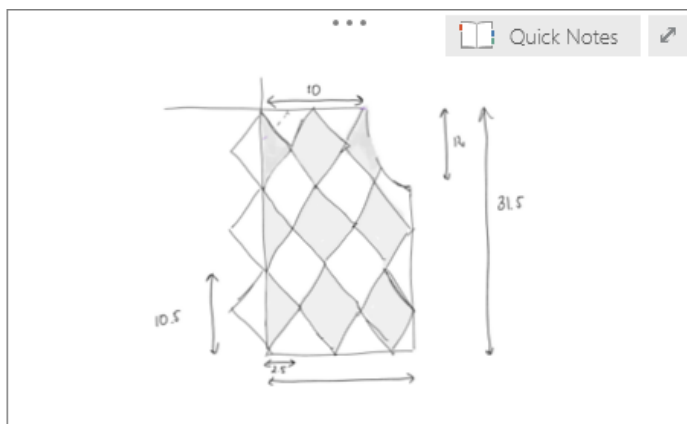
Integrate with Outlook

- Take notes on Outlook or Lync meetings
- Insert meeting details
- Add Outlook tasks from OneNote



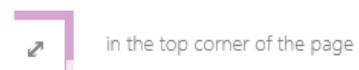
Add Excel spreadsheets

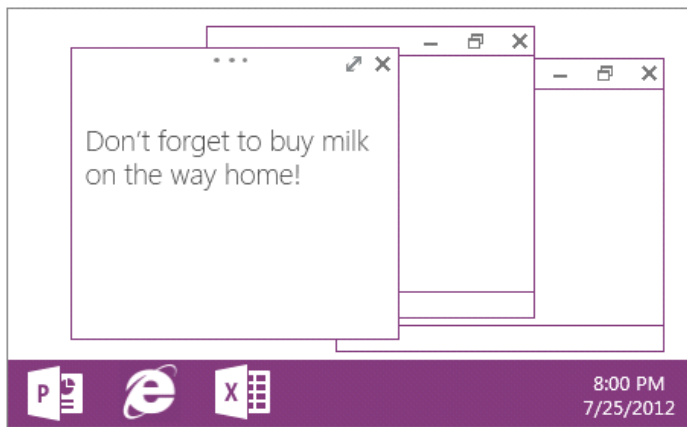
- Track finances, budgets, & more
- Preview updates on the page



Brainstorm without clutter

- Hide everything but the essentials
- Extra space to focus on your notes





Take quick notes

- Quickly jot down thoughts and ideas
- They go into your Quick Notes section



in your taskbar

OR



+ N on your keyboard

Corporate Overview

Monday, September 21, 2020 11:12 AM

Jobs

- Fed
 - Fed - helping them directly
 - Fed projects - doing their projects
- Us
 - Legacy dev - spring and pivotal?
 - New contracts
 - R&D
 - Infra - implem focus, maintenance/patching, security
- Overseas too in sing, canada, etc

Projects

- Document Registry using S3
- Doc Storage using Filestack
- Text engine for converting pics to pdfs (ML)
- Key/ identity management
- Expert Gateway - creating cases to be reviewed
- Doc exchange for shareholders or business transactions
- Notification engine
- SPF suite of startup kits
- Doc classification- sorting claims/ cases
- Sensitive info identification and redaction (ML, PII)
- Data broker- batch and on demand

“So... tell me about Maximus” Tips

- We are a professional services firm
- We work primarily with government agencies at federal, state, and local levels focused on health and human services solutions
- More recently, we won some significant bids to assist with contact tracing efforts
- Although the corporate headquarters is in the DC area, we have operations all over the US and about 8 other countries abroad
- We hire interns and recent college graduates from a wide range of majors (Social Sciences, Government, Business, STEM, Liberal Arts, etc.), leveraging experiences both in and outside of the traditional classroom environment.
- Our selection process emphasizes career readiness skills such as leadership, teamwork, critical thinking, communication, and work ethic.

Selling points:

1. We are very proud of our mission: Helping Government Serve The People
 - a. “Making a difference when it matters most”
2. Strong balance sheet: maintained interns & full-time hires despite pandemic
 - a. 2020 Top 100 Internship program
 - b. Able to get new work during the pandemic
3. Opportunities for professional growth
 - a. Emphasis on promoting from within
 - b. Online portal for professional development opportunities for employees

Meeting Notes

Thursday, November 12, 2020 3:05 PM

- 11/12 COVID readi - product to help get vaccines (vaccination)
 - PrepMod
 - Consent forms for vaccines- search for clinics, select clinic, select form, fill out info, screen, get consent
 - Covid Readi - vaccine getting vaccination
 - Pre-register (consumers info), provider sign up, clinic sign up
 - To find the people, business, facilities to actually administer vaccine as fast as possible
 - Helps businesses sign up and facilitate process to apply to state to become a vaccine provider
- 11/13 IMR Invoice system
 - Current physical pdfs, one packet per ca, copy for them to download
 - CA ids, working functionality already with IMR portal or whatever
 - Needs
 - Notification CA invoices ready and **reminder** if not downloaded/ notif if downloaded
 - Repl what exists with NOARFIs and IMR Portal
 - Time downloaded, by what user
 - Release after new year
- 11/25 Moveit
 - Frontend Sketch/mockup for invoice download/ view interface (CAs) - the user story
 - Then figure out backend
 - Frontend Sketch for finance team - user story
 - Same
 - Figure out libraries (aws sdk etc, s3 etc)
 - Any questions glaring
- 6/28
 - Kubern- for docker images
 - Codeshuttle - library for deploying repos
 - Used by jenkins (via launchpad file) to deploy updates to repo
 - Launchpad - uses config var (just a hashmap) to set settings of deploy
 - Can change type of container (jnl, beefy, helm)
 - Autodeploy - deploy to dev after build? Says yes automatically
 - Prop.debug.global
 - cdPipeline function
 - ◆ Runs groovy
 - ◆ Can set endpoint (ci/cd/etc), type, type (pipeline)
 - ◆ Runs buildConfig (Map func returns map) - used by all pipeline files
 - ◆ Runs pipelineRunner on buildconfig
 - Basically library of interfaces and classes that set vars and deploy based on what is in files and what configurations r determined based on that
 - Jenkins - can manage repos and watch builds, see debug of job processing, change launchpad file
 - When jenkins detects launchpad file in repo, will show up in jenkins then will run jobs based on configs
- 6/29 Pair Programming with George groundCtl
 - ./dgroundctl restart -u [namespace]
 - To restart jenkins instance then build and deploy in mongo
 - Use instance.conf file to play around with env, helm, etc
 - ./dgroundctl -l -u [namespace] for log
- 7/2 Codeshuttle Demo
 - FedQic implementation with SPF
 - Source code, compile, build, unit test then make reports
 - Qic repo is for infrastructure billing and app deployment
 - Through Spf, Ansible
 - From dev injected to template, codeshuttle handles infra and deployment of app
 - Does risk assessments to make sure aws resources not being deleted
 - Aws config report - non prod and prod have compliance issues that are calculated
 - 2 hours to standup, uses RDS (Postgres & SQL), 400 GB for all volumes
 - SPF additions/ integrations
 - Deployprep script in spf - tells info to developers (config.deployprep.enabled = true)
 - Use autoversioning (github with package.json) in build pipeline will auto assign #, then change in github (but don't run pipe this time)
 - Can force ami changes (such as for patching) - through terraform or jenkins
 - Cloudtrails splunk dashboard
 - Api to see every aws api call and see errors for specific people
 - All updates - sees all updates (create, delete etc) by any onelogin user in any maximus region, can select per user
 - All errors - sees everything in cloud causing errors
 - All data in s3 buckets
 - Deployment interface enhancements
 - Interface created from pipeline after config
 - Has endpoint (default, CD,), repo, env, version
 - Can choose from a few env- has protections if things havent already been deployed to env

7/6 MongoDB - through rest api () - json object

Make a dashboard to show deployments and information

- Next Tues
- Deployment Name
- Deployment Status

7/14 Helm upgrade Jenkins- iterate and work through it, well documented, should be easy... right?

- Upgrade helm template for jenkins for 3.0.0 since the helm team doesn't want to maintain it

7/16 meeting with David

- Ask about anything from hr - nothing mark needs to send me over, last thing

- Ask about Splunk
- Ask about dashboard/ api link

- **Ask about sec clearance**

7/22 Codeshuttle Dashboard

- Persona driven - login, based on group gives diff display
 - Should be able to modify
 - Based on login - could be static then based on login
 - Roles: Developer (details), Overall(fame/shame, who sucks, dials and percents), Consumer, **what kind of roles**
- Shows KPIs - columns
 - Job status
 - % failed/success
 - Code coverage trends
 - Versions deployed in env - everything has a version :'(
 - Test results
 - Vulnerability trends
 - Build times
 - Build Velocity
 - Last change
 - Issues/troubleshooting - java compile, pod error
 - Starters/approvers
 - Env creation time/age
 - Patch versions/ami versions
 - **Probs have as expanded row or click on job and new page**
- Data
 - Jenkins data (mongo api)
 - Sonarcube
 - Jenkins api
 - Gradle api
 - Svn api (stat api)
 - Github api
 - Cherwell
 - Traefik- ingress to kube, proxy for front and backend
 - **How to get data**
 - Traefik - <http://codeshuttle-traefik.se.maximus.com:8080/api/providers/kubernetes/backends>
 - Not part of fedic, records kube activity (backend activity) for codeshuttle
 - Fedqic - <https://fedqic.codeshuttle.maximus.com/api/>, <https://fedqic.codeshuttle.maximus.com/api/state?filter='repoName':'qic'>
 - Basically where everything is, combines all- same thing as codeshuttle
 - ◆ Within fedqic has jenkins, github, etc.
 - ◆ Maximus codeshuttle seems to have same format but slightly different info
 - With state, date of snaps, places deploy, last success job
 - User, version for each env, date time, replay,
 - Jenkins
 - Need service acct name, token
 - Start here: <https://fedqic.codeshuttle.maximus.com/jenkins/script>, run command = "cat /var/jenkins_home/jobs/token.txt"

```
println "[cmd] ${command}"
proc = command.execute()
def sout = new StringBuffer()
def serr = new StringBuffer()
proc.consumeProcessOutput(sout, serr)
proc.waitForOrKill(1000)
println "[cmd.out] ${sout}"
println "[cmd.err] ${serr}"
```
 - Then take [11baba4a8534a71652a4d07f9601fa281d], go to postman
 - ◆ At <https://fedqic.codeshuttle.maximus.com/jenkins/crumbissuer/api/json> go to Auth and put username SVC_CodeShuttle and password [token] - receive crumb
 - ◆ At <https://fedqic.codeshuttle.maximus.com/jenkins/api/json> in headers put in jenkins-Crumb with value [crumb]
 - Pipelines.api/json is the top level, has urls under jobs[].url
 - ◆ Also has health report[0].description for string, [0].score for score (/100)
 - ◆ Each url from there has health report in same format, jobs[0] url which has info on builds, buildable, and can go into different builds using diff url

- Low users (30 max)

- Eventually

- Form dashboard will be able to restart job or something - basically take over jenkins dashboard

- To do:

- Masked passwords
- Connect props to retain username and password info
- Infrastructure for pages based on role
- Use real data
 - Add user, date, replay, datepicker
 - Add version
 - Structure - normal is version but some are deployed > [env] > version where env is dev, uat, prod
- Make a wireframe - should be easier after seeing data
- Need to ask for kubernetes access (above)
- How to get into jenkins, sonarcube
- What is this <http://maximus-training-dev.se.maximus.com/api/snapshot>

- Questions

- Cors.... Oh cors....
- What is diff codeshuttle and fedqic- just 2 versions of same thing?
- The hell is with this html

8/18 Karl Splunk meeting

index="max_inventory" sourcetype="softwareVersions" | dedup ipaddress, package | table hostname ipaddress package version _time

- Looks for software running things we don't want like old versions - returns table with software version, hostname, ip address, time stamp from max_inventory table
- Can make dashboard, edit source code with xml to run query - better than putting in query ourselves
 - In splunk app for infrastructure
 - Look for centos6 hosts

- o Pie chart by OS version, then each OS by version
- 8/19 Secret Meeting
- Currently done through CMD line- ground control
 - o Use gcontrol create customer - builds a repo
 - During creation- with secrets in json, encrypt through cmd line- **secrets encrypt [file]** to encrypt to BPE
 - Go to secrets/customer- under codeshuttle, add this new repo
 - o In groundcontrol - secrets edit system - look for your secret repo name - change repo url to new git url, change secrets key from output of create customer
 - o Run script from that input
 - o To apply secrets- dgroundcontrol secrets apply customer -u [config/plconf/maximus-shared-env/prod-tools-cluster/prod/maximus]
 - o Take 2
 - Create dummy repo - name must match customer name (ex. Texaseb_secrets) (private)
 - In groundcontrol - ./dgroundctl pull all
 - ./dgroundctl secrets create texaseb
 - o Takes us to secrets file for editing - wq
 - o Run script /tmp/dgroundctl-cs and write commit message
 - Apply wit
 - Purpose -
 - o Rn its basically using github and scripts to act as a secrets store holding encrypted data
 - o Ideal - build an interface that can... do that...? Should be able to take an input json/file
 - Store somewhere, should be compatibles with
 - o Type of secret, whats the id, whats the token, description - hit apply and inserts data into json structure - and ultimately saved to git repo - clone, edit, push
 - Like in groundcontrol/lib/secrets/secrets/create
 - Multiline text field private field
 - File uplaod
 - file paste box
 - o Editing/ deleting - all managed in git
 - o Eks - gp2 volume , add encrypted volume
 - o -r jquery is good for stripping quotes
 - Next-- cyberark (company standard)
- 8/23 Carls stuff
- Need a markdown fixer for the CAB markdown documents - can be made into diagrams as per env runbook
 - Want to take in json, combine with template the matching keys, put in config merge to merge them, then spit out markdown - then that goes to codeshuttle as html
 - o Starts with target, reverse engineers template, then can validate or whatever u know
 - The point of this is to make better data entry - all it does if have a form to accept data then send it to the right place so the pipeline can funnel (github)
 - Lansweeper- ton of information
 - Dynamic create template and json - from the target markdown - possibly run through config merge ourselves
 - Search plus query, inside chart takes base search- search name is ct
 - o Use base option to take results from search and do other stuff
 - o Use id param with search to take results of old query (temp file)
 - o Set default
 - o Title for panel all events
 - o Title for table - total = num
 - o Date/time
- 8/23 more work on jenkins board
- Highs and criticals over time to show a leaderboard- for see saws/ fixed, not fixed etx
- 8/24 David's Disney logs/ File Explorer
- App
 - o Front end doesn't matter (dynamic html) - maybe python dhtml2
 - o Backend - query account servers to go to lambda to access directly - string back to interface
 - o Dockerized - front end needs http service (python, apache, OICD auth) on Kube acct
 - o Logs for ec2s, for any logged into accounts- query by acct or by OS, log in and go find var/logs, logs
 - Refresh automatically - cat log, tail log, etc
 - Backend query to python - ssh to server, list directory everything in **var logs**, can also do things like **config logs** with tomcat, **op directory**
 - o Server logs for everything in acct
 - o Filter out destructive places
 - o SSH from slave to node in another account - aws accounts
 - Evaluate if you clicked directory - ssh again with directory path into ls and give tree
 - o If file ssh still and do tail to tail logs
 - Explore ssh over ssm execute (everything), solve for linux and windows (doesn't do ssh, fileshare)
 - Write code to connect over ssm to execute on ls
 - o Ssm seems too hard to implement- ssh
 - Display data in listed tree, other is tail or cat display - and connect periminko? Ssh library or ss
 - Auto refresh
 - Step 1- run locally and try to ls
 - o Need access to maximus-shared-env to deploy websites too
 - o Python obj with boto3, cat/tail, searches, etc
 - Point is to allow people to see things without giving them ssh access - limiting while showing logs, ssh for them
 - Ask Brad during standup
- 8/26 Fedqic
- Talk to hari on what you want to see
 - o What devs want - choose thing like Fedqic, detailed info
 - Talk to David
 - o What proj people want - time series, velocity (num records, num times deployed/built), indicators- how many successful, how many failed
 - o Trend in api/project, only last 10 days tho - mongo
 - o How to pull from test, one of these pipelines have that
 - o Put thing in project - if thing was aborted or not,
 - o Make a box and whisper per day, of fail/success
 - Could do bar charts, etc
- 9/2 Secrets Interface
- Add fields for all things in file except type maybe
 - Process character turn to \n for files and rsa:
 - Prompt for customer name - name for github repo - must
 - Dashes to underscores (only in json)
 - Need to take returned key, url, repo branch so it can interface with system secrets (repo to get into all customer secret repos)
 - Way to manage security -
- 9/3 - logs
- Backend - connect to ssh
 - o Paraminko, parallel ssh (performance), netmiko, carl/Montana
 - o 2 diff docker for windows - target, source
 - Source - volume to manipulate, use python simple http server -
 - Target - what we're investigating
 - Step 1
 - o get docker to spin up with hello world on 80 (source)
 - Step 2
 - o One docker to another with key over port 22 through library above (reliable and simple)
 - Step 3

- Can I send command through browser to make library connect to target
 - Then run command to get data back to browser (like a dir) cus it works like api
- Step 4
 - Then interface

9/7 logs

- Livelogs
 - Dockerfile - loads a bunch of stuff, art, cgi (run python), assyncssh (ssh lib), copy over ssh keys
 - Copy from ssh directory to user/lib/cgi/bin/id-rsa but doesn't work (?)
 - Does stuff so cgi will work after that
 - Exposes port 80, set work dir,
- Keys - in ssh
- Target server
 - Add user, sets pass as test, copy over ssh config, start ssh, open port 22, uses service as daemon
- Code
 - Front end - this actually makes sense!
- Cgi-bin - backend
 - Can change name in docker compose 25,26
 - Builds and puts in

9/14

- Frontend function calls backend to retrieve key from backend which generates key (based on name convention)
- Style
 - Dropdown that gives regions (<https://jqueryui.com/selectmenu/>)
 - Remove dots from list
 - Movable center line - make the skinny button and maybe an arrow
 - Tab later - much better openable
- On click - choose server & go get key (write to drive), then use key to ssh and navigate within server

9/17

- How to navigate loaded ssh var log
 - Function after load that turns all anchor clicks to load mainNav

9/21

- Other links in odeck dashboard:
 - Jenkins
 - Sonarqube
 - Risk of EOL

9/22

- Port 22 - no policy
- No user - someone changed something, no spf
- No key - no spfServer
- Add top/bottom of page
- Ryans notes
 - Spacing on li,
 - prevent back button?,
 - More easter eggs,
 - dashboard view (health)
 - Current open sessions on box,
 - Clickable path to go higher
- Demo stuff
 - Inv navbar
 - At beginning -
 - Start of visibility platform (Observation deck)
 - Want to provide the functionality to view info such as logs and configs on any machine in an acct
 - Eventually more things like EOL info, Jenkins data, sonarqube data, etc
 - (This is for texas eb)
 - The idea would be to log into Odeck- so it can record what accts have access to
 - Click region (for this acct)
 - Displays servers currently running in texas-eb env (shows name of each ec2)
 - Can click on any - displays u01 partition on server, can navigate within server
 - Apache-> Readme
 - Currently text based logs, tars
 - U01/ansible/ansible/cache/ tar
 - Up to root, var -> log-> Dmsg
 - Other visibilities
 - Server see running services, cpu/ mem utilization
 - Currently in dev env, plan to rollout next release
 - Only supports linux, on scope to add windows and maybe docker

9/30 Fedqic etc

- Versions deployed is good
 - Want to see when version was deployed
- Don't hardcode stuff - make group for account based not for service base

10/5

- Giles backbutton input
- Suss unauth after long time- ssh token/ creds timeout?

10/19 Hari meeting

- Windows machines
- Test info
- Jenkins
- Sonarqube
- Invite Hari to George's - DONE
- AWS Inventory - ec2, rds, whats patched and id for it
 - Filter for each project
 - Config message logs - Ansible messages
- Get thigs like CPU, mem, drivespace, etc from AWS
- List services/ processes on a server

11/5

- Demo stuff
 - Usability things
 - Logout
 - Url path
 - Back and forward navigation
 - Clickable path
 - All of contextenu
 - Rename - no duplicates
 - Path wont work
 - Copy - can put path and new name
 - Cant already exist
 - Move - enter dest folder
 - Must be new folder
 - Delete
 - After confirmation
 - Download
 - Download be download
- Part of securing maxE integration - added usability for devs in QA without requiring the use of ssh
- Load if

- o No tabs yet
 - o Mismatch url server and tab server
 - o Mismatch url path and tab path
- Don't load if
 - o Page load
 - o Same server and path
 - o Error page and url server matches
- A & (B | ((C | D) & (E & !C)))
 - o C | D & E & !C
- git config --global credential.helper store

- Top frame - approved colors
- No main nav border, no footer
- Actual picture profile from login or wherever
- Light theme switchable

12/3

- Right align numbers
 - o Folder bold
 - o
- Monospacing
- T rex game
- Df / h
 - o Top processes, disk space, cpu mem
- Like ls -la in terminal

12/6 Hari Meeting

- Windows - done
- Linux - in the works
- Heap Dump - interesting
- Top processes - hit box
- AWS Resources
 - o Separate pages
 - In own tab - add more info, more dashboardy things
 - o RRD tool or use as javascript
- Loans against 401k
 - o zamn
- Tensorflow
 - o AI/ML big things

12/10

- f

Command Line Notes

Saturday, August 15, 2020 10:53 AM

. Means current direct

.. Means parent

/ means root directory

~ means home directory

Right click paste

Can use a/b to look/create/change to b inside of a

Use quotes on files names of \ for escape spaces

Tab completion

pwd - print working directory

ls - list files

Ls <nameoffolder>	to look inside folders
Ls -l	long output (permissions)
Ls -a	all files and directories
Ls *<str>	List all end with str
Ls <str>	List all begin with <str>

cd - go to highest

cd ..	go to parent
cd ~	Go to home direct
cd -	go to back to last
cd /	Go to Root direct

Cp - copy

Cp <file> <dir>	Copy into dir
Cp <file> <dir>/<newfilename>	Copy and change name
Cp -R <dir> <newdirname>	Copy dir & contents (tree)
Cp -R <dir>/	Copy dir contents (leaves)

Mv - move

Mv <thing> <target>

Rm - remove

Rm -R <dir>	Recursive delete dir
-------------	----------------------

Mkdir - make directory

Mkdir -p a/b	Make parents if needed
--------------	------------------------

Touch - create empty file

Echo - output

Glob - tries to find any

Expand - tries all alternatives

*	Match any number of chars - empty does all
<str>	Match where str in file names
<str><str2>	Has str in it, ends in str2
<str><str2>*	Has str followed by str2

?<str>	Matches _<str>
[<char>]	Match any enclosed char (can have mult options)
[char - char]	Can get chars between (alpha or nums) can do both
{str, str2}	Match any enclosed str (can have mult)
{char..char}	Does it for all between char

SSH, Subnets, CIDR

Wednesday, August 26, 2020 3:47 PM

Secure Shell

- How computers connect to servers- they use private half (on computer) with public half (on the server)
- Ssh-keygen -C <comment> -f <where to put it>
 - Creates ssh key
 - Has passphrase (another layer of security)
 - Creates 2 files with the private and public halves (.pub is public)
- Can go on aws into ec2 server -in key pairs can import your public key
 - Remember use security groups, add address for anywhere (ssh is port 22)
- To enter into ec2, need ip address (goes into putty) and private key file (in putty security -> auth), then it will ask for passphrase

Subnets

- Instead of all computers being able to talk to each other, put computer on subnet, and subnet can attach to big net and receive info but big net cannot necessarily access subnets
- OSI Model - layers of protocols
 - HTTP for apps
 - TCP (transport control) for transport
 - IP (internet protocol) for network - deals with IP addresses, Routing, Subnetting
- IP addresses (4 and 6, 4 is more common)
 - Unique address for each computer (4 from 0 to 255) - still not enough (4.3 billion)
 - First num defines network (and last 3 were host)
 - Then changed to more creativity with defining class (by range on first num) and then each class has diff nums to define network (Class A has 1 num network, Class B has 2, etc)
 - Use subnets - they connect to big net through gateway
 - More control

CIDR

- Would that be enough IP addresses? Do we even want to give all of them public IP addresses
- Classless Inter-Domain Routing - has the IP address then /num where num represents the number of network bits (10.0.1.128/25 says 25 of the 32 bits are for the network and the last 7 are for potential hosts) - more flexibility
- Still only 4.3 ip4 billion ip addresses (ip6 has 3.4×10^{38} different ip addresses)

CLI

Wednesday, September 16, 2020 12:43 PM

- Go into IAM and after downloading aws cli get access keys
- Aws configure asks for keys
- Make bucket - `aws s3 mb <s3://bucketname>`
- Copy to s3 - `aws s3 mb <bucketdns>`
- Can configure profiles with `aws configure --profile <nameprofile>`

Git

Wednesday, September 16, 2020 10:42 AM

Git Basics

- To start- git init - creates Git repo
- Has .git repo which has info on repo
- git status - which branch
- git add - prepare for commit, add files and dir/
- git commit - need set with username and email if first time (--global command)
 - Brings up vim - wants a description
 - Hit I (for insert) then message
 - To leave - esc, :, wq - for write and quit
 - Or use commit -m "message" to avoid vim
- Git log

Branches

- Create then switch branches
 - Create with git branch -c <namebranch>
 - Get on with git checkout <namebranch>
- Git branch shows branches with current highlighted
- Don't forget the head (where git is pointing) - use git checkout to switch head to current
- To create and switch - git checkout -b <namenewbranch>
- Stash or commit changes before changing branches
 - Git stash - stashed all changes
 - Git stash list - to see them when you come back
 - Git stash pop - to bring changes back into working directory

Git history

- Git log - shows head pointing to what then the history of commits
- Git log <ref> - shows history before ref point (e.g. first-branch or commit ID)
- Git log --all
- Git log --author, git log --since <datestring>, git log --until <datestring>
- Git log --oneline - simple version
- Git log --graph - for graphical view of commits
- Git show <commitID> - used for detailed info on commit such as changes
- Git diff - lots of options for difference two points
 - Git diff - differences between working directory and head
 - Git diff <commitID> - differences between working dir and commit
 - Git diff --cached - what changes added but not committed
 - Git diff <commit1>..<commit2> - diff between commits (can do with refs also)

Theory

- Branch - indep line of dev, Tag - specific point in time on branch, checkout - go to branch, commit, push, workspace - where repo is on computer, untracked files, working area - changed files, staging area - ready for commit, local repo, remote repo - on GitLab

GitLab

Thursday, September 24, 2020 10:49 AM

Intro

- Easy connect github repo and gitlab, give it a name, url, description, private/public
- GitLab helps you build, deploy and configure your app
- Version control, mult independent branches, fast distributed, open source and standard
- Features - provides comprehensive DevOps lifecycle support, improves cycle time, reduces cost and time to market
- Workflow cycle
 - Create code - push code - build and test with push changes then more build and test - reviewed and approved - merge with master - then one last build and test
- Benefits - one tool, can run distributed jobs on separate machines, and in parallel, and optimize delivery

Flow - building workflow

- Use merge requests (solve merge conflicts), can review code before add, test beforehand
- Git add (add to staging), git commit (staging to local repo), git push (local becomes master)
- https://docs.gitlab.com/ee/topics/gitlab_flow.html

GitFlow

- Git flow init will create git repo, base branch and creates branching scheme
- Git flow feature start <namebranch> creates new feature branch
- Git add . To add current workspace to staging
- Git commit -m "message"
- Git flow feature finish <namebranch> to merge feature branch
- Git flow release start <version release> to create release branch

Namespaces & Groups

- Namespace - a group, subgroup, user
- Group - group... shared resources, can make groups within groups and allow convo up/down the group
 - Set visibility (public, internal, private), access rights of users within a group
- SSH key - can easily create for a profile (profile - ssh key - get commands to create in CLI)

WebIDE - Pretty helpful, make merge requests, see diffs in browser

Markdown

- Easily create styles
- # is for heading 1, ## for subheader, ### for smaller
- [term] (url) is how to insert a link
- List with - bullets, can do ordered lists with 1. 2. or 1. 1. will change it
- Dash - [] will create a checkbox for todos
- / and list of quick actions

GitLab Pages - Basic web page deployment for demos?

Kubernetes - clusters of nodes, Gitlab works with Google cloud

- Works with auto DevOps, creates cont deployment pipeline

Jenkins- For containerization

- Jenkins Pipeline allows put git repo in (credentials possibly), the build branch, build triggers (periodic build, hook triggers ?), build (fresh builds or no), configuration (data tree for docker file, docker registry, cloud, images of build)

DevOps CI/CD

Thursday, September 24, 2020 1:57 PM

Lifecycle

- Development
 - Plan, code, review, build, test, repeat
- Operations
 - Deploy, operate, monitor, release
- Sync between the 2
- Patterns
 - Devops of dev, app delivery, ongon support (operations)
 - DevSecOps - Security can be added
- Applying Agile - steer (business planning), build (create and test intensely), deploy tested things in cont mechanism and review, operate (optimize, monitor)
- CloudOps Agile - business driver aligns business solution with cloud services, then create in agile mechanism, powered by DevOps strategies

Continuous Delivery

- Process - auto prep code for release to production, from source control, build, stage to production
- CI is the source control and building, CDelivery is that plus staging and production, Cdeploy is no waiting to deploy
- Benefits - automates software release processes, better dev productivity (don't worry about deploy), faster bug detection, quicker update delivery
 - For devs - better end to end visibility (easier find changes), improved quality (iterative), faster feedback loops, integrated compliance/ sec, less dependency on operations
 - For Ops - helps innovate for transformation, stable and available environment, better operations throughput, less bottlenecks
 - For QA - integration QA in dev, constantly ready to deploy apps, early detection/resolution, faster rollbacks

CDelivery Principles

	Repeatable and reliable process for deployment	Do difficult tasks more often
	Every member is responsible for something in releasing	Maintain everything in a source control
•	Automate everything	Define "done" as released
	Facilitate target quality metrics	Implement Continuous Improvement
•	Foundation of Cdelivery <ul style="list-style-type: none">◦ Configuration management - ensure assets are known/ tracked◦ Continuous Integration - devs deploy code to repo freq◦ Continuous Testing - software testing early, often, and automatically	
•	CD Pipeline for CloudOps - Hypothesize what will be best, Build it, Measure the effectiveness, Learn and repeat <ul style="list-style-type: none">◦ Process - Fetch release → deploy Canary →cutover manual approvals → deploy production → tear down canary → destroy older deployment	
•	Right Practices - automated tests, use same build throughout pipeline, fix problems in code (not in place), give up problematic code	

CDelivery Elements

- Architecture (non functional aspects for operations)
 - Users, have apps which might use things like DBs
 - Change so users use dispatcher which selects original app or new modules, each using DBs
 - Don't have monolithic app, use new modules and dispatcher to separate things appropriately
- Culture - culture and arch versus velocity (change)
 - Start with Maintainers (traditional IT), then Optimizers (enable Cloud), Innovators (accelerate)
- Patterns (workflows/ procedures) in CI
 - Build packages only once
 - Deploy similarly in every environment (don't change too much)
 - Use them smoke tests - works, available, has config, dependencies
 - Ensure all production environments similar
- Patterns in CD
 - Commit, either Accept commits or test, stage while capacity test while integration test, then production

CDelivery tools

- Fabric - delivery infrastructure (ex Kubernetes)
 - Python library, has CLI tools, SSH, suite for depolyment and System admins
- Capistrano (framework for writing scripts)
 - Remote multi server delivery, in Ruby, extends Rake (domain specific language)
- Jenkins (open source automation server for dev)
 - Java, integrate 3rd party platforms, great for large scale, good with cloud and CloudOps
 - Sample CD in Jenkins- load balancers, regular repo things
 - Set up
 - Manage jenkins - global tool configuration - needs some installs
 - Maven, JDK installations, Git, Gradle, Mercurial, Ant, Docker
 - With whichever used to deploy, can set configuration - build triggers, source code, build settings, post steps

CI Principles

- Benefits - Increase visibility with more communication, catch issues early, helps debugging, builds solid foundation (automation), Immediate response of does it work, reduce integration problems
- Principles - Single repo, automate builds, build commits on integration machines (not original), lots and lots of testing, enable builds to do self-testing, fast builds, simplify getting latest versions, ensure visibility, automated deployments
- Applying CI in pipeline - verify code quality (testing, scanning, dependency), package (containerize, Maven), Release (Canary, CD - auto)
- CI in CloudOps - dev environment - has server, sync, engagement and quality environment (same 3), the 2 attached by a console

CI Practices

- Check in frequently, nice code, test code, don't check to broken code, wait until system builds
- Step by step - use private workspace, commit change to repo, CI server can monitor those changes, then build/run/integrate, CI server deploys artifacts for testing, make sure its versioned, builds successfully/not, fix issues, continue to integrate
- In Cloud - Zuul server has scheduler and server will schedule and run with git Daemon

Goals CI - Valid builds, auto tests, integrate changes to main

Goals CDep - auto release, faster feedback loops, manage changes in production pipeline

Goals Cdel - more CI, release planning, manual deployment

Cloud Continuous Delivery - need app management (orchestration), quality management, Client approval, and proper interface to connect them

Repos & Pipelines

- Bitbucket - for repo management and piping - similar to github repo, but pipelining too
 - Has bitbucket-pipeline.yml file which describe how to deploy, then choose how to deploy (configuration file)
 - Setup CI with Node.js - package.json is the file for dependencies
 - With CD pipeline - go to deployments - uses bitbucket-pipeline.yml which needs deploy configs info
- 3 types - work (models, projects, execution), master (security, versioning), execution (code is deployed)
- Best productivity - development (dev and test, not secure info) and production repos (sensitive, user data, user facing)
- Selection criteria - supported version controls, team size, release schedule/ plan, project size, external integrations

CD Workflows

- Atlassian tools - Kanban board is for issue creation (backlog), assign tasks, create branch, choose type (bug fix, etc), from branch, name

Branching Strategies

- Usually have Dev/master branch, QA for quality assurance, staging branch, UAT branch, Production for end user
- Production based - focus on prod
 - Prod branch is default, features branch from prod, QA has completed features (test branch), feature branch merges w/prod
- Int Environment - adding features together first then to prod, not one at a time
- Feature branch - focus on features, testing during feature prod, rebase production to features

CDelivery Maturity Models

- Base - essentials work on own - build script and build machine, deployment scripts, some testing, tool gen reporting
- Beginner - self service builds, nightly, build artifacts stored, mostly standardized deploys but to 1st stage, test at build, team level reporting
- Intermediate - build on commit, repo of dependencies, configuration, standard builds, deploy to test/prod, auto test nightly, cross silo
- Advanced - triggered builds, build clusters, DB deploys, multi tier deploys, security, risk mitigation, report trending
- Extreme - VM snapshots builds, gated commits, containerization, cont deployment to prod, 100% code testing, cross silo analysis

How to automate app release

- Need orchestration and packaging, dependencies (what for what), approval requirements, CI/CD methodology, auto release automatically
- Benefits - streamlined processes, reduced timelines, less manual, more agility/flexibility, improved productivity, collaboration, risk manage
- Batch release with AWS- in updates (after review, test, approval) when scheduled, deployed, rebooted, but in security - run scan, review report, approve patches, create patch schedule, deploy patches

Docker

Friday, October 23, 2020 9:39 AM

VMs

- Run code in fully isolated, indep environments - virtual hardware done by software
- Architecture- Infrastructure (hardware), then host OS, then Hypervisor (runs VM, type 1 uses hardware, type 2 uses host OS), guest OS(s) (controlled by hypervisor to run each app), each has binaries and libraries, then the app src code
- Use cases - web hosting to separate customers, test environ to match production servers
 - Testing- use VM running Docker for example
- VMs are houses (more not less) and Docker are apts (sizeable)

Docker Basics

- Containers are isolated processes- faster, easier to use
- Architecture- infrastructure, Host OS that runs Docker, Docker Daemon (runs Docker in background), binaries and libraries (as Docker images), then app
- No guest OS (super bulky mem and disk resources)
- Use cases- Run single process in isolation such as app components

Toolbox-

- Docker CE/ EE- comes with Daemon and CLI
 - Daemon- Runs on Linux, used by Rest API then that used by Docker CLI
 - Client runs, uses Docker host (daemon) which connects to containers and images (**remote dameon**), and that connects to Docker registry
 - Docker Enterprise Edition- certified images/plugins, docker datacenter, vulnerability scans, same day support (as Docker images), then app
 - Docker Comm Edition- production ready, open source
 - 2 release channels - edge (releases every month) and stable (3 months, easier to maintain) - predictable
- Docker Compose
- Docker Machine
- VirtualBox- load Linux in Mac OS/ Windows - type 2 hypervisor
 - Used by Docker Toolbox -> Docker Machine to create servers and thus VirtualBox VM with Docker install
- Docker Quickstart Terminal- creates server with VM and Docker via Docker Toolbox
- Kitematic- graphical tool to manage containers and Docker images
- Docker for Mac/ Windows
 - Newer, no VBox, uses **hypervisor type 1**, installs CE, Compose, Machine and Daemon runs **locally**
 - Requires version 10+ for either Mac/Windows but for VBox easier, can use any terminal, no native Windows
 - Might be a lil slower
- Images vs containers
 - Image- File system + parameters (single package)- No state, doesn't change, can download/build/run an image
 - Container- The act of running an image (instance of class), **immutable (ALL changes lost)**
 - Are managed images (e.g. hello world) that live in docker registry (holds docker repos (collection of versioned images) and each image has tags) in docker hub

Builds

- Run container, change things, commit changes
- Use Dockerfile- blueprint for creating docker image

Dockerfile

- Starts with FROM - use a base image (other docker image, usually official) syntax **ImageName : tag**
- RUN basically allows commands (e.g. mkdir, pip install, etc)
- Period (.) still means current dir
- WORKDIR sets working directory inside container
- COPY to copy files from this folder to docker container copy at the end cus docker can cache (save time on unchanged)
 - Usually copy dependencies separately
- CMD run but for running instance, not build (which is when RUN occurs)
- Can log in to docker hub with **docker login** then push image with **docker image tag <name> <username>/<reponame>**
- Can delete with docker image rm <name or image ID>
- Run container - **docker container run -it** (for ctr C) **-p 5000:5000** (for ports) **-e FLASK_APP=app.py** (env variable) **web1**
 - Can run detached (-d flag)
- Come with logs- docker container logs <name> - (-f for real time)

Code Changes with Volumes

- To modify edits to app.py, can build and run again or use -v then path to files
- Flask volume - **\$PWD:/app** while named volume is **web2_redis:/data** (name not file path)
- Build with **docker image build -t <namebuild>** .

Docker Networks

- Internal- LAN - local area network- on same router with router ip address similarities
- External - WAN - wide area network
- Run a redis container- **docker container run --rm -itd -p 6379:6379 --name redis redis:3.2-alpine**
- Run flask app- **docker container run --rm -itd -p 5000:5000 -e FLASK_APP=app.py -e FLASK_DEBUG=1 --name web2 -v \$PWD:/app web2**
- Create bridge network and auto do DNS - **docker network create --driver bridge firstnetwork**
- Open redis CLI - **wintpy docker exec -it redis redis-cli** then check keys **KEYS *** then ctrl d to stop
- Bridge driver can only connect over 1 docker host

Persist Docker data - Use a named volume

Sharing data between containers- Just add --volume-from <container> and in Dockerfile have line VOLUME["path/to/folder"]

Optimize Docker Images

- Docker Ignore file- certain files get omitted from version control (environment files, private files) with .dockerignore ext
- First line is .dockerignore (ignore theses), can include directory with .example/, can ignore all with extension with ****/*.*.py** and can create exceptions with !special.txt
- Can make it a whitelist and not a blacklist by setting first line to * then next is .dockerignore
- Can put dependencies in docker file but means rebuilds and more complex files

Docker Scripts

- ENTRYPOINT can run project codes with just 1 dockerfile (many sh files in package)
- Make sure RUN chmod to give the script permissions
- CMD passes thing to ENTRYPOINT instruction, default entrypoint is bin/sh -c

Cleanup

- Use ls on images, containers, docker system df
- Docker system prune (with -f force or -a auto for infreq accessed)
- Stop all containers- docker container stop \$(docker container ls -a -q)

Compose

- Uses YML, has version, services (name them whatever) then within them have images, ports, build (. For dockerfile in current directory)
 - Versions- 1 is legacy (doesn't have services), 2 is single service and mostly similar but 3 is best
- Lists in YML have - prefix
- Depends_on so service only executes when other services done
- For env variables use env_file then make list of files - with env variables declared
- Docker-compose build - build your docker compose.yml
- Docker-compose pull - pull down image
- Docker-compose up - scales runnable services
- All 3 - docker-compose up --build

General

- Be stateless, use env variables, make redis ur friend, 12 steps to portability

Spring

Monday, November 9, 2020 1:38 PM

Intro

- MVC Controller
- Annotations- to mark to java and us
 - @SpringBootApplication mark main class- includes @configuration,@enableautoconfiguration, @componentscan
 - @EnableAutoConfiguration - looks for auto-config beans on classpath
 - @Autowired to let java wire, use with @Qualifier to specify, @Required for bean setter methods
 - Conditional Configs
 - @ConditionalOnClass & @ConditionalOnMissingClass - spring use auto config bean if class is present/absent
 - @ConditionalOnBean & @ConditionalOnMissingBean -
 - @ConditionalOnProperty, @ConditionalOnResource, @ConditionalOnWebApplication, @ConditionalOnNotWebApplication
 - @ConditionalExpression and @Conditional for more complex conditionals (expression or class)
- Uses a lot of dependencies (in semi html format)
- Spring Boot Starters - to cut down on dependencies
 - Spring MVC
 - Web starter, test starter, Mail, Data JPA (persistence)
- Super Modular -
 - Data access (JDBC - abstract layer DB, ORM - integration layer object relation (virtual DB), OXM - abstract for XML, JMS)
 - Web (WebSocket, Portlet, MVC)- bunch of MVC stuff
 - Core (Beans, Context, SpEL) - fundamental parts of the framework, implementation (Bean), Context (further builds)
 - Tests

Spring Bean

- Object instantiated, assembled by Spring InversionOfControl Container (object defines dependencies without creating)- represents an instance maybe?
 - IoC helps by managing/retrieving dependencies without us manually doing
 - Instance of AnnotationConfigApplicationContext Class to build up a container of a beans config class

Spring Recs

- Don't use default package, declare package in classes instead
- Put main class in base package, package by feature is preferred way - create subpackages for each to go with main package

Spring Boot Actuator (dependency) <https://www.baeldung.com/spring-boot-actuators>

- Actuators bring **monitoring** features (metrics, traffic, info)
- Comes with most endpoints disabled- only available are /health and /info
 - Lots of endpoints - for monitoring different things
- Health Groups

Endpoints <https://docs.spring.io/spring-boot/docs/1.5.x/reference/html/production-ready-endpoints.html>

- ffff

Properties

- @PropertySource to add property source files "classpath:foo.properties"
- Can also be done with XML <property-placeholder>, YAML files, command line
- Use @value to inject a property
- More granular test control @TestPropertySource, @ConfigurationProperties property hierarchies

Coding Beans https://www.tutorialspoint.com/spring/spring_ioc_containers.htm

- ApplicationContext and ClassPathXmlApplicationContext - loads bean config file **bean.xml** and takes care of creating objects
- getBean - uses bean ID to return generic object
- Bean config file - XML that glues classes together - under src directory, used to create beans with IDs and control creation (independent)
 - Beans config metadata used to make properties for each def
 - Class - attribute is mandatory and specifies class to create bean (simple class file), Name, Scope, Constructor, properties, mode (autowiring/ inject or lazy), initialization method, destruction method
 - Scope - new bean each time one is needed - prototype, one per container- singleton, per HTTP request - request, http session-session, global-session
 - Singleton uses cache after creating bean
 - Lifecycle - initialization method - **afterPropertiesSet** method from interface **InitializingBean** for java and **init-method** for XML
 - Destroy method in interface **DisposableBean** for Java and destroy-method attribute for XML (when creating bean, then ref method by name, can also set methods as default-init-method for example)
- IoC container - creates objects, wire them, configure, life cycle
 - Uses config metadata (XML, java annotations, java code)
 - 2 containers - BeanFactory - simple, basic DI while ApplicationContext is mor/better/specific functionality such as event listeners
 - Can autowire relationships between beans in XML with byName attribute, byType, by constructor method, constructor args
 - Preferred is to explicit wire
- BeanPostProcessor
 - Another interface for more logic to do before/ after initialization
 - Can also use RegisterShutdownHook to ensure graceful shutdown - uses AbstractApplicationContext to find xml with beans config
- Can define child and bean parents (inheritance), uses parent = "" by bean id, can also create templates for beans (don't use class = , use abstract = true)
- Can make inner beans, can pass lists/plural values to

Testing

- @RunWith(SpringRunner.class) is for using JUnit
- @DataJpaTest is for testing persistence layer (h2/ db, Hibernate, @EntityScan, SQL logging)
- TestEntityManager to setup DB records
- @MockBean stands up a temp bean kinda, bypasses actual layer (service, persistence, etc)
- @WebMvc testing for spring mvc, @SpringBootTest for integration testing

New Hiring

Tuesday, June 29, 2021

1:33 PM

Onboarding

- Github
 - Need license and access
 - Make a Personal Access Token (allow whatever permissions you want), authorize sso (located in settings, developer settings)
- Trello
 - Need account and access to tools engineering
- Workspaces - need workspaces
 - SPF - need to run spf workspace setup - <https://devops.pages.dev.maxird.com/spf-help-site/general/setup/aws-workspace-setup.html>
 - WKS setup
 - git clone <https://github.com/maximus-codeshuttle/wks-setup.git>
 - Go in and run ./setup_nix.sh
 - For each of the 9 options y or n (mostly yes except 5-7 options)
 - Keep trying and don't be afraid to edit setup_nix.sh

How to prepare workstation

- Codeshuttle- don't use it but pretty good knowledge - put code in space
 - Consistent, cont way, testing, etc , quickly make variants based on infrastructure
 - Uses spf to call on to build aws stuff
 - Eventually- feedback loop - visibility to see state of process, where it failed, etc
- Aws workspace - 2 weeks
- Onelogin - need it
- Groundctrl- half a day sucked but good for downloading everything
- Jenkins - what it is and how to use it, getting your own instance
- Getting your own pod
- Github PATs - need it to contribute code

Ansible & YAML

Friday, July 2, 2021 3:56 PM

- Intro
 - Tool for IT automation by Red Hat
 - Runs YAML in Ansible Playbooks
 - Connects to nodes and pushes out small programs (modules)
 - Ansible executes modules (over SSH) then removes them
 - Lots of integrations (aws, atlassian, splunk, cisco, google cloud, etc)
 - <https://training.galaxyproject.org/archive/2019-06-01/topics/admin/tutorials/ansible/tutorial.html#what-is-ansible>
- How it works
 - Control node (master) uses managed nodes, connecting to them and creates tasks
 - Task - a call to a small program (module) to and how to call it (config)
 - Ansible does a lot of the work- written in YAML to reflect desired state
 - Playbooks - YAML that decides how to use modules (tasks) - applied to group of hosts
 - Host files - Ansible spec file defined groups of hosts (servers)
 - Galaxy
 - Tool to retrieve roles from Ansible, is a repo
 - Role - folder containing tasks, template, files, def values for vars
- YAML
 - Basically just a dict
 - <https://www.cloudbees.com/blog/yaml-tutorial-everything-you-need-get-started>

Docker

Thursday, July 1, 2021 9:55 AM

VMs

- Run code in fully isolated, indep environments - virtual hardware done at software
- Architecture- Infrastructure (hardware), then host OS, then Hypervisor (runs VM, type 1 uses hardware, type 2 uses host OS), guest OS(s) (controlled by hypervisor to run each app), each has binaries and libraries, then the app src code
- Use cases - web hosting to separate customers, test environ to match production servers
 - Testing- use VM running Docker for example
- VMs are houses (more not less) and Docker are apts (sizeable)

Docker Basics

- Containers are isolated processes- faster, easier to use
- Architecture- Infrastructure, Host OS that runs Docker, Docker Daemon (runs Docker in background), binaries and libraries (as Docker images), then app
- No guest OS (super bulky mem and disk resources)
- Use cases- Run single process in isolation such as app components

Toolbox-

- Docker CE/ EE- comes with Daemon and CLI
 - Daemon- Runs on Linux, used by Rest API then that used by Docker CLI
 - Client runs, uses Docker host (daemon) which connects to containers and images (**remote dameon**), and that connects to Docker registry
 - Docker Enterprise Edition- certified images/plugins, docker datacenter, vulnerability scans, same day support
 - Docker Comm Edition- production ready, open source
 - 2 release channels - edge (releases every month) and stable (3 months, easier to maintain)- predictable
- Docker Compose
- Docker Machine
- VirtualBox- load Linux in Mac OS/ Windows - type 2 hypervisor
 - Used by Docker Toolbox -> Docker Machine to create servers and thus VirtualBox VM with Docker install
- Docker Quickstart Terminal- creates server with VM and Docker via Docker Toolbox
- Kitematic- graphical tool to manage containers and Docker images
- Docker for Mac/ Windows
 - Newer, no VBox, uses **hypervisor type 1**, installs CE, Compose, Machine and Daemon runs **locally**
 - Requires version 10+ for either Mac/Windows but for VBox easier, can use any terminal, no native Windows
 - Might be a lil slower
- Images vs containers
 - Image- File system + parameters (single package) - No state, doesn't change, can download/build/run an image
 - Container- The act of running an image (instance of class), **immutable (ALL changes lost)**
 - Are managed images (e.g. hello world) that live in docker registry (holds docker repos (collection of versioned images) and each image has tags) in docker hub

Builds

- Run container, change things, commit changes
- Use Dockerfile- blueprint for creating docker image

Dockerfile

- Starts with FROM - use a base image (other docker image, usually official) syntax **ImageName : tag**
- RUN basically allows commands (e.g. mkdir, pip install, etc)
- Period (.) still means current dir
- WORKDIR sets working directory inside container
- COPY to copy files from this folder to docker container copy at the end cus docker can cache (save time on unchanged)
 - Usually copy dependencies separately
- CMD run but for running instance, not build (which is when RUN occurs)
- Can log in to docker hub with **docker login** then push image with **docker image tag <name> <username>/<reponame>**
- Can delete with docker image rm <name or image ID>
- Run container - **docker container run -it** (for ctr C) **-p 5000:5000** (for ports) **-e FLASK_APP=app.py** (env variable) **web1**
 - Can run detached (-d flag)
- Come with logs- docker container logs <name> - (-f for real time)

Code Changes with Volumes

- To modify edits to app.py, can build and run again or use -v then path to files
- Flask volume - **\$PWD:/app** while named volume is **web2_redis:/data** (name not file path)
- Build with **docker image build -t <name>build** .

Docker Networks

- Internal- LAN - local area network- on same router with router ip address similarities
- External - WAN - wide area network
- Run a redis container- **docker container run --rm -itd -p 6379:6379 --name redis3.2-alpine**
- Run flask app- **docker container run --rm -itd -p 5000:5000 -e FLASK_APP=app.py -e FLASK_DEBUG=1 --name web2 -v \$PWD:/app web2**
- Create bridge network and auto do DNS - **docker network create --driver bridge firstnetwork**
- Open redis CLI - **winpty docker exec -it redis redis-cli** then check keys **KEYS *** then ctrl d to stop
- Bridge driver can only connect over 1 docker host

Persist Docker data - Use a named volume

Sharing data between containers - Just add --volume-from <container> and in Dockerfile have line VOLUME["path/to/folder"]

Optimize Docker Images

- Docker Ignore file- certain files get omitted from version control (environment files, private files) with .dockerignore ext
- First line is .dockerignore (ignore theses), can include directory with .example/, can ignore all with extension with ****/*.py** and can create exceptions with **!special.txt**
- Can make it a whitelist and not a blacklist by setting first line to ***** then next is .dockerignore
- Can put dependencies in docker file but means rebuilds and more complex files

Docker Scripts

- ENTRYPOINT can run project codes with just 1 dockerfile (many sh files in package)
- Make sure RUN chmod to give the script permissions
- CMD passes thing to ENTRYPOINT instruction, default entryptoint is bin/sh -c

Cleanup

- Use ls on images, containers, docker system df
- Docker system prune (with -f force or -a auto for infreq accessed)
- Stop all containers- docker container stop \$(docker container ls -a -q)

Compose

- Uses YML, has version, services (name them whatever) then within them have images, ports, build (. For dockerfile in current directory)
 - Versions- 1 is legacy (doesn't have services), 2 is single service and mostly similar but 3 is best
- Lists in YML have - prefix
- Depends_ on so service only executes when other services done
- For env variables use env_ file then make list of files - with env variables declared
- Docker-compose build - build your docker compose.yml
- Docker-compose pull - pull down image
- Docker-compose up - scales runnable services
- All 3 - docker-compose up --build

General

- Be stateless, use env variables, make redis ur friend, 12 steps to portability

Go (Golang) & Groovy

Wednesday, June 30, 2021 10:24 AM

Go

- Intro
 - How should I put this... maybe like python java baby but more python and someone went through the work of rewriting the whole syntax
 - <https://productcoalition.com/reasons-why-golang-is-better-than-other-programming-languages-4714082bb1b1>
 - Efficient computing, execution and ease of writing
- Code - <https://tour.golang.org/>
 - Like python but different syntax, kinda hype a little oversimplified
 - Imports - use factored way- all imports in between ()
 - Can import math
 - Var **[Var_name] [type]** for explicit decl,
 - Vars have defaults (0, false, "")
 - Can cast as usual
 - Use := assignment to not have to declare type
 - Loops
 - For is same but init and incrementor are optional (basically a while loop)
 - Slicing
 - Can slice a slice and can extend the slice on the previous slice - go has capacity which changes when slicing inside another slice, wont change if on same slice

Groovy

- Intro
 - Like a Java Python baby but more java
 - OO language, class based
- Setup
 - Should come with wks setup script but can download from <http://www.groovy-lang.org/download.html>
 - Can run in vscode - use Code Runner Extension
- Hello World example

```
class Example {
    static void main(String[] args) {
        // Using a simple println statement to print output to the console
        println('Hello World');
    }
}
```
- Auto import util, lang, etc
- Variables - java data types and non-defined yet (**def**)
- Same as java - operators, loops, if then, break continue, writing methods
- Files - can do lots of input output but very different than java or python syntax - https://www.tutorialspoint.com/groovy/groovy_file_io.htm
- Strings, numbers - similar java but for numbers groovy uses wrapper classes to make the literals (values) into classes (process called boxing) and turned back for say operations (unboxing)
- Cool ranges -
 - 1..10 - An example of an inclusive Range
 - 1..<10 - An example of an exclusive Range
 - 'a'..'x' - Ranges can also consist of characters
 - 10..1 - Ranges can also be in descending order
 - 'x'..'a' - Ranges can also consist of characters and be in descending order.
- Lists like python (**[]**) while maps are similar (**[:]**)
- Dates/ times similar to java,
- Random
 - Static variables - can be changed in a function
 - X ?: y means return x if not null otherwise return y

Helm

Tuesday, July 6, 2021 11:31 AM

- Intro
 - Package manager for Kube- like yum or apt
 - Useful for reducing complexity, enabling automation
- How it works
 - Used by installing Helm on client machines in Kube cluster
 - Deploys charts- like a packaged app with version, pre config resources, deployments
 - Written in YAML, uses Tiller (helm server) to be installed on **cluster** (with Helm)
 - Receives requests from client
 - Single package to deploy to cluster - yay!
 - After Helm installed, can use to get apps
 - MongoDB, MySQL, others and put into Kube cluster with **helm install __**
 - Helm Charts
 - YAML manifests combined into one package - makes installing things on nodes in cluster easier as its all 1 - so the actual node configs
 - Helm releases specific instances of nodes on cluster with info such as version, pre config resources, deployment info
 - Repo - group of published charts which can be made available to others
- Why its good
 - Easy deployments and resource finding for entire cluster(helm install __)
 - Build chart once and use over and over
- Commands
 - Helm create [CHART_NAME]
 - Has .helmignore - like gitignore is files ignore when packaging
 - Chart.yaml - info about specific chart such as version num, etc
 - Values.yaml - values to put into templates
 - Charts (folder) - stores charts, might call a Chart from a Chart
 - Teamplates (folder) -actual YAML manifest deploying with chart (to cluster)
 - Used for other Yaml like deployment.yaml, service.yaml, config.yaml

Jenkins

Wednesday, June 30, 2021 3:25 PM

- Intro
 - <https://www.tutorialspoint.com/jenkins/index.htm>
 - Jenkins is a CI platform tool that helps manage cont. build code (not deploy)
 - Can auto perform builds on code changes as well as perform tests and send notifications
- Setup
 - Download - <https://www.jenkins.io/download/>
 - Java - For idk java stuff geesh all of this is java
 - Apache Tomcat - Jenkins does build on Tomcat server, needs space to perform builds and save archives
 - Github - Cus it builds repos
 - Maven - also project management, used for managing java based projects
- Use
 - Create jobs - configured by project type (maven, multi-config, external build), make jobs details
 - Can execute windows batch commands from job setup
 - Can see output after job creation
 - Can run on code commit or on merge or manually
 - Nice because can see history and things
 - Testing- Comes with Junit, other plugins can be installed
 - Reporting, Notifications

Kubernetes

Wednesday, June 30, 2021 4:12 PM

- Intro
 - Kube is a cluster orch system - manages nodes and containerized apps
 - Containers package software so apps can be released and updated without downtime
 - <https://kubernetes.io/docs/tutorials/kubernetes-basics/>
 - Great for cloud native apps - apps with small, specialized components easy to manipulate
- Basics
 - Kube coordinates highly available clusters of connected computers, giving them a containerized app
 - Scheduling and distribution across cluster in efficient way (unlike before with integration to host system - less mobile)
 - Needs volumes (abstraction to persist data - normally data dies with container deletion)
 - Has control plane (coord cluster) and nodes (run app)
 - Control - schedule, maintain state, scale, rolling out updates
 - Node - VM or physical worker - has a kubelet (agent for managing node and communicating with control), also has tools to handle containers such as containerd or Docker (usually at least 3 nodes)
 - Command Line Interface is **kubectl**
 - Runs on client (kubectl), and server (Kubernetes version on master)
 - Get cluster info, can get nodes for hosting apps
- App Deployment
 - To run containerized app on running Kube cluster, need deployment config (to deploy)
 - During creation need container image and # replicas
 - Once running, Deployment Controller monitors app instances and creates new nodes if the node hosting pod goes down
 - Can setup proxy with **kubectl proxy --port=8000** and **curl http://localhost:8000/**
 - Add version at the end to get kubectl version
 - Add api to end to start using kubectl api
 - Api server has endpoints for each pod, need pod name then can curl
 - ◻ **Curl** <http://localhost:8000/api/v1/namespaces/default/pods/PODNAME>
- In app
 - Kube pods- created when app instance is deployed, represents one of more app containers and shared resources (volumes, ip address, info on how to run containers)
 - Node - worker machine (see above), runs at least kubelet (to talk with master) and container runtime (docker, etc to do work on images)
 - Can have many pods on node
 - Kubectl
 - Can get pods (name, age, status, # running), can describe pods (ip address, ports, lifecycle events, start time, status, containers, etc), get logs (requests, uptime,), exec commands on container (as opposed to pods),
 - **Kubectl exec -ti (POD_NAME) -- bash** to start a bash session
 - ◻ From here can access other things like app port (8080 on localhost)
- Public apps
 - Pods have lifecycles, lost if node (worker) dies - usually Replicaset to counter
 - Each pod has own IP address, need way for frontend to auto reconcile
 - Services- routes traffic across pods- allows for pods to die and replicate from info
 - Defined in yaml or json
 - Pods targeted set by LabelSelector- without it wont create endpoints, good if user wants to manually map service to specific endpoints (or using external)
 - Allow pods to expose each unique IP
 - Types
 - ◻ ClusterIP (default) - exposes Service on internal IP in cluster (service can only be accessed from cluster)
 - ◻ NodePort - exposes on same port of each Node (using NAT) - now accessible from outside cluster
 - ◻ LoadBalancer - external LB in cloud and external, fixed IP to Service (superset of NodePort)
 - ◻ ExternalName - maps Service to contents of externalName field through cname record
 - Use Labels - key value pairs attached to objects - for designating object env (prod, test, dev), for versioning, classif
 - Has unique cluster IP of **internalPort:ExternalIP/Protocol**
 - Can use describe to get service info
- Running Mult App instances
 - For maintaining availability and low times by scheduling appropriately
 - Uses endpoints, can do rolling updates without downtime
 - 1st instance is counted as replica
 - **kubectl Get deployments** - name, ready (current inst/desired), up-to-date (num repl to get to desired), available, age
 - **kubectl get rs** - get replicaset - info on name (normal name plus random string), desired (# replicas)
 - **kubectl scale deployments/[NAME] --replicas=[NUM]** to scale to NUM replicas
 - **kubectl get pods** will show 4 pods now each unique name and IP -o wide for extra info
 - **kubectl describe deployments/[NAME]** will show details of a deployment
 - **kubectl describe services/[NAME]** will show details of a deployment's services - has an IP, port, endpoints
- Updating instances
 - Rolling updates - incremental update pods with new ones - no downtime because updates done on available nodes
 - ◻ Allows for rollback to previous, shows CI/CD with no downtime
 - **kubectl set image deployments/[NAME] [NAME]=[NEW_IMAGE]** to update to new image
 - **kubectl rollout status deployments/[NAME]** to see status of rollout
 - **kubectl rollout undo deployments/[NAME]** to go back one step
- Tutorial <https://cloud.google.com/kubernetes-engine/docs/tutorials/hello-app>
 - Gcloud command line tool
 - Create a container
 - ◻ git clone <https://github.com/GoogleCloudPlatform/kubernetes-engine-samples> gets dockerfile
 - ◻ **docker build -t us-east1-docker.pkg.dev/\${PROJECT_ID}/hello-repo/hello-app:v1** . Runs dockerfile
 - Puts it in local env and calls it hello-app (v1), in us-east1 regional host for artifact registry
 - ◻ Can test with **docker run --rm -p 8080:8080 us-east1-docker.pkg.dev/\${PROJECT_ID}/hello-repo/hello-app:v1** or by opening new terminal and typing **curl http://localhost:8080**
 - Put it in artifact registry
 - ◻ Use command line to authenticate to registry, then push (same as run command but push not run)
 - Create kluster
 - ◻ Set region with **gcloud config set compute/region COMPUTE_REGION**
 - ◻ Then run **gcloud container clusters create-auto hello-cluster**
 - Puts a cluster (with nodes) into your region
 - Deploy app to cluster
 - ◻ **Dep1 - kubectl create deployment hello-app --image=REGION-docker.pkg.dev/\${PROJECT_ID}/hello-repo/hello-app:v1**
 - ◻ **Scale - kubectl scale deployment hello-app --replicas=3**
 - ◻ Can test with **HorPodAutoscaler - kubectl autoscale deployment hello-app --cpu-percent=80 --min=1 --max=5**
 - When cpu on node is 80% scale out to a max of 5 and min of 1
 - Expose to internet - Service!
 - ◻ **Service - kubectl expose deployment hello-app --name=hello-app-service --type=LoadBalancer --port 80 --target-port 8080**
 - ◻ Use kubectl get service to see where your lb is (external IP) and can go directly there in browser
 - To update image
 - ◻ **Update docker image - docker build -t REGION-docker.pkg.dev/\${PROJECT_ID}/hello-repo/hello-app:v2 .**
 - ◻ **Put in artifactory - docker push REGION-docker.pkg.dev/\${PROJECT_ID}/hello-repo/hello-app:v2**
 - ◻ **Apply new image as rolling update - kubectl set image deployment/hello-app hello-app=REGION-docker.pkg.dev/\${PROJECT_ID}/hello-repo/hello-app:v2**
 - To tear it down
 - ◻ **Service - kubectl delete service hello-app-service**
 - ◻ **Cluster - gcloud container clusters delete hello-cluster --zone REGION**
 - ◻ **Images- gcloud artifacts docker images delete REGION-docker.pkg.dev/\${PROJECT_ID}/hello-repo/hello-app:v1 --delete-tags --quiet and then the same for v2 - gcloud artifacts docker images delete REGION-docker.pkg.dev/\${PROJECT_ID}/hello-repo/hello-app:v2 --delete-tags --quiet**

Kube 2

Wednesday, July 7, 2021 2:17 PM

- <https://learn.acloud.guru/course/kubernetes-deep-dive/learn/k8s-bp/ccaad6f5-69bc-5d29-5658-f76c7a3aaf5f/watch>
- Basics 2
 - Kube rests between VM/ hardware (cloud stuff) and cloud native apps (w/specialized components that are easy to change/scale)
 - Kube itself has control plane (1+ nodes) and workers (lots of nodes)
 - Nodes are instances (VM/cloud/physical hardware)
 - Control plane - Kube API, scheduler, controllers, persistent store
 - Etcd - stateful part of plane, persistent store, problematic at scale
 - Api - where most traffic is, grand central for cluster
 - Kubectl talks to, sends requests, makes way to worker nodes to do
 - Can have namespace - sub cluster basically
 - Api
 - Everything in Kube is defined here (pods, deployments, etc)
 - Restful, uses http to do CRUDs (create, read, update, delete) generally using (GET, POST, PATCH, UPDATE, PUT, DELETE)
 - User uses kubectl, sends YAML which is desired state of app then
 - Declarative config - with kubectl, request that state be a way and cus it observes
 - Apps thus have doc and always up to date
 - Core API group ("") - pos and SVC, apps group (deploy, replicasets), auth group (rback), storage group
 - Special Investor Group - people who look after a group
 - Features
 - Alpha - fairly unstable, Beta - taking shape, GA- ready for prod
 - Objects
 - Pod - smallest unit of scheduling/ managing (contains containers/apps)
 - Deploy- Wrap around pods, have job of scaling, updates
 - DaemonSets - one pod per node
 - Stateful Sets - for pods with state reqs
 - Secrets, Volumes, APIs etc
 - App Architecture
 - Made of web service (page/ thing), with persisting back end and maybe load balancer to handle load, back end storage, potentially secrets (values passed between front and backend)
 - Can containerize front end, backend into two different pods, use service as LB, use deployment to wrap front end, secrets for secrets, persistent volume for persistent volume
 - Can all be externally accessed, all are things on the cluster
 - Uses Docker images (YAML code) to spin up resources that will result in desired state
 - Uses Cloud integration to develop LB (also YAML)
 - Networking
 - All services (in front of nodes) are endpoints so have Ips
 - Need dynamic DNS to help them find each other
 - Modern networking is dynamic, scalable, real time
 - In YAML - can specify resource by kind (service, deployment), give it metadata (names, labels), spec (ports, type (LB)) etc
 - Basics
 - Nodes can talk, Pods can talk without Nat, every pod has an IP address
 - Usually has 443 (HTTPS), big flat network where every pod can see others
 - Services
 - Stabilize network - cus pods are ephemeral (don't persist) - service is able to traffic to new pods with new pods and not old dead pods
 - Front end - each service has name, IP (never change), pods pre-config to find service
 - Back end - label selector - choose which pods to traffic to, uses endpoint object
 - Endpoint object - has list of Ips and pods to see which pods have selector and are alive
 - Types - all basically stable networking point
 - LB - integrates with cloud LB
 - ClusterIP - default - cluster gets IP only accessible from within cluster
 - NodePort - enable access outside cluster - cluster wide port, attach to any node IP
 - Can set manually with flag but default 30000 - 32767
 - Good for querying api for example
 - Pod Network, Node Network, and Service Network
 - Service Network doesn't have interfaces or anything, talk by **kube-proxy** daemon set runs on node which writes tables that redirect traffic to pod network
 - Proxy is in charge of local networking on node
 - Proxy iptables mode - doesn't scale well, not best for load balancing
 - Proxy IPVS mode - stable, uses Linux kernel IP virtual Server, Layer 4 LB
 - Super simple to provision things - just kubectl apply -f .\{file_name\} to apply a manifest
 - Recap
 - Node network- all nodes communicate (Kubelet to API server and pack)
 - Pod Network - 3rd party implemented - big and flat, IP per pod
 - All containers in pod share pod details, can talk to each other with localhost
 - Ephemeral (so don't go to them directly)
 - Services - to front end has IP which is stable, to backend uses selectors to decide where to send traffic
 - Has list of healthy pods (ep-endpoint list)
 - 3 types, up to cloud LB
 - Not really a network, just captures traffic and reroutes to pod network
- Storage
 - Kube volumes - decoupled storage
 - Can share pods between volumes
 - File and block storage
 - Standards based, persistent, big API
 - PersistentVol - actual storage (ex. 20G hold)
 - PVCClaim - voucher to use PV
 - StorageClass - good implementation
 - Setup - cloud/whatever -> Container storage interface -> kuber pers volumes
 - CSI - out of tree - good cus closed source, not dep on kubernetes updates,
 - Volumes exist on cluster indep from nodes
 - Made with yaml files, pvc also
 - Specs for PV and PVC **must match**
 - Can set storage, set disk by name
 - Access modes - read write once (one pod), readwrite many, readonlymany - one claim (pod) at a time
 - Only some volumes do some modes
 - Reclaim policy - delete (kill on pod release)/ retain (don't kill, manual delete)
 - Dynamic prov/ storage classes- dynamic is better
 - For connecting to backend services through plugin, defined by YAML file
 - Portworx/ storageOS, aws ebs for example (provisioner)
 - Give a bunch of params as well - name (in metadata) most important as pvc uses (pod uses pvc name also)
 - Use annotations to set default
 - Will create for us
 - Default storage class will be created even if yaml file doesn't specify pvc
- Coding
 - Code
 - Into repo/ directory maybe in svn/github
 - Docker
 - Put into Docker file, run docker build
 - Builds docker image, usually on docker cloud
 - Wrap in Kube deployment
 - Kube
 - Use kubectl (& yaml) to config and deploy
 - In yaml specify want to use that container image (need repo and name)
- Deployments
 - Need pods (wrap around)
 - Technically wraps around replicaset which manages pods
 - Runs one type of pod exactly, can make replicas, cant make pods with diff specs
 - Can config update in strategy (e.g. rolling) in yaml
 - Labels - used to identify which pods are in deployment (metadata run), and to tell deployment which pods are in the group (matchLabels run)
 - Change the yaml file don't do it manually (even though rollbacks super easy)
- Scaling
 - Horiz pod autoscaler - creates more pods (for when pods full)
 - Create like any other Kind of Kube object, needs specs, name, replicas
 - Can set target CPU for example
 - Deployment needs resource requests and resource limits
 - Request how much of a cpu you want to begin with
 - For knowing when to scale, tells deploy, actually does
 - Autoscaling/v1 only supports scaling based on cpu but
 - Cluster autoscaler - creates more nodes (for when nodes full)
- RBAC (role based access control) and admin control
 - Api auth control - first check who is it, then are they allowed, then modify and valid request (e.g. must have label or image must come from repo), validates scheme/ object
 - AuthN - client has creds with request, check yes/no by giving to external system
 - In form of - Bearer tokens, client certs, bootstrap tokens, externals
 - No users in Kube tho - manage outside cluster with IAM, active directory
 - System comp have service accts- managed by kube, can manage things like how many service accts
 - Authz - **deny by default** - j
 - Mode - how to authz, min Node (kubeleets to API), RBAC (which users which actions which resources)
 - When creating cluster user gets god permissions
 - Instead use
 - roles - resource in api, has rules - verbs (e.g. get, list, watch), resources (e.g. pods), api group (e.g. apps)
 - Role binding- the who for this, defined namespace, user (or group)
 - ClusterRole/ClusterRoleBinding - not restricted to namespace
 - Admin Control - can have multiple
- Other stuff
 - DaemonSets - only one pod per node
 - StatefulSets - manage scale stateful bits of app
 - Jobs - api object for running spec number of pods and complete
 - CronJob - run jobs on a schedule
 - PodSecPolicy - happens after all other stuff, to make sure pod cant do stuff to

underlying computer for example

- Pod Resource Requests/ limits - request cpu/mem feeds into scaling, etc
- Resource quotas - namespaces and quotas - set limits on resource usage
- Custom Resource - can create cluster than use that in other clusters

Postman

Tuesday, July 6, 2021 11:44 AM

- Intro
 - Tool for designing, managing and testing APIs
 - Lots of templates and features to help get started
 - Intro to APIs
 - Have to set server to return requests
 - Server can accept get, post, put, patch, delete, view, head, copy etc
 - Server returns code, body based on what is sent
 - Has versions too
- Can send apis
 - Create mock server, create your request(s), create mock server, change env to this newly built server, go into a request, send it and see if returns response body
- Can create APIs
 - Under Define tab - to define how api behaves - postman supports OpenAPI, RAML, GraphQL
 - After that can create collections to test things
 - Develop tab - for building out - can create mock servers, docs (markdown), env.s
 - Test tab - test collection versions validated against schema
 - Observe tab - monitor performance of API (specific schema and version perf)
- Testing
 - Can send a request with top bar to any url, then receive body, status code and headers which can be used after
- Collections
 - Can create variables within collections
 - Can use saved variables in collection requests (as param) with syntax {{VAR_NAME}}
- Env Variables- similar to collection vars but bigger scope?
- Global vars - assume biggest scope
- Can script tests pretty easily but syntax is kinda wonky
 - It's just javascript- <https://learning.postman.com/docs/writing-scripts/test-scripts/>
 - Can run test for entire collection
 - Can loop/ set work flow with `pm.setNextRequest('request_name')`, (name is under collection and tab at top of page)
- Can also do pre req scripts (scripts run before req)

SPF

Tuesday, July 6, 2021 11:29 AM

<https://devops.pages.dev.maxird.com/spf-help-site/videos/series/quickstart.html>

- SPF quick start - <https://devops.pages.dev.maxird.com/spf-help-site/commands/spf-quick-start/>
 - Can run spf-quick-start to build new project quickly
 - Creates file structure with services.yml - includes deployment in nodeJS
 - -g for generate (lambda, ec2 (docker), fargate (container), rds, account, bare (just network))
 - -o for output folder, -d (public domain in aws), -n network, -r region, -a spf acct in aws
 - To run
 - Source awsinfo.sh (usually in keys folder), test with aws s3 ls or something else using aws
 - Spf-gen (works with spf project to create terraform)
 - Create-roles script creates ssh keys
 - Link aws ssh key pair with your key
 - Terraform init (load modules), terraform plan (--compact-warning), terraform apply --auto-approve, terraform destroy --force to tear down
- Running Samples - <https://devops.pages.dev.maxird.com/spf-help-site/general/articles/samples.html>
 - Made in particular account and domain (change)
 - Change to run yourself
 - project.yml - project name, system (name), siem_name (secureworks)
 - Settings.yml - terraform bucket (spf acct prefix followed by terraform remote state), region (write in unless you want us-east-1), public domain
- Catalog Format
 - Physical model - virtual hardware
 - Resources
 - Type - lots of options, describes which aws to use basically
 - Documentation
 - Services - List name of services live on resource
 - Zone_override - if diff zone than service, such as public AZ for service but more secure for resource
 - All come with base definitions
 - Logical model - software to run
 - Services.yml - has services at top of doc
 - Zone - req by services (management, data, app, public, external)
 - Like type of service
 - Uses - when services USES another service
 - Documentation - write stuff
 - Alias - alias (another service, inherits rules from another service) or rules (how communicate port/protocol)
 - Http, https, oracle, mysql, etc inheritable services available
 - Top files
 - Meta.yml - lists environments- only dev by default
 - Logical.yml - points to which files should be in spf project
 - Dev folder - has layout towards top meaning actual resources
 - Each resource has services- to define which services live under resource (usually 1:1 but maybe more (lbs.yml))
 - Layout.yml - offers pointers to files
- Deployments
 - How software is run, config after provisioning, under Services
 - Toolchain property- coding lang to be used by deploy (e.g. python, java, ruby, node, etc)

- Lambda
 - Usually detach resource from vpc ()
 - Bundle code & supporting in zip file or have generator bundle for us using inline
 - Inline property- code in yaml or ref import that will run lambda
 - Handler - function run when lambda invoked (function inside file name- e.g. index.handler)
 - Runtime - language used
- User-data deployments - start-ups for ec2
 - Need sub_type - ami (bash for centos ami, posh for windows, cloud-config for cloud)
 - Docker - config docker daemon
 - Content - define unix/cloud config/ posh commands
- Ecs
 - Define images in ecs- how many, how much CPU/mem, env vars, name, image, cpu, mem, release version, desired replicas
- S3 content
 - Source- path to website content, can make target file where to define, cache-control

Other

Thursday, July 1, 2021 9:42 AM

- Ansible
 - Tool for IT automation by Red Hat
 - Runs YAML in Ansible Playbooks
 - Connects to nodes and pushes out small programs (modules)
 - Ansible executes modules (over SSH) then removes them
 - Lots of integrations (aws, atlassian, splunk, cisco, google cloud, etc)
 - How it works
 - Control node (master) uses managed nodes, connecting to them and sending small programs (modules)
 - Ansible does a lot of the work- written in YAML to reflect desired state
 - Playbooks - YAML that decides how to use modules (tasks)
- Build tools
 - Such as Maven, Docker, Ant, etc where some are based on xml, some on json. Etc
 - Does work to gen source code, gen documentation, compile source code, JAR compiled code, install Jar in a repo
- Chef
 - Config management- good for CD, quick bug detect and fix, cloud integration
 - Written in Ruby and Erlang
 - Pull config
 - Components
 - Workstation - admin- creates recipes (plural cookbooks) and uses Knife on Command Line to upload cookbooks
 - Server - where books stored, can be local or remote
 - Node - Ohai is on node, fetches current state, Chef client comm w/server, nodes can be diff
 - Types
 - Solo - no server, cookbooks live on nodes
 - Hosted Chef - server on cloud
 - Chef Client - traditional workstation, server, node
 - Private Chef - server hosted within enterprise
 - Code
 - Within repo(directory) has cookbook folder
 - **Chef generate cookbook [NAME]**
 - ◻ Creates tree with lots of stuff and recipes folder (initially default.rb is only recipe file)
 - For server - can use manage.chef.io, download chef starter kit, move unzipped kit to cookbooks folder in repo
 - ◻ **Knife cookbook upload [book_name]** to upload cookbook to server, connects server and workstation
 - ◻ On node side - need IP of node (**ifconfig** works), then back on workstation **run knife bootstrap [IP] -ssh-user [username] -ssh-password [password] --node-name [name of node]** - now configured
 - ◻ To run book on node, go on manage.io and edit run list, add recipe to run list, run chef-client service in root of node (may have to sign in with **su**) to update - then book should be running
- Domain Specific Language
 - A programming language created to do a very small set of tasks
 - An example would be SQL which can only do DB operations- can't make a calc or whatever (bottle opener)
 - Not an example - Java - can make a lot of diff programs like calc, map, visuals (hammer)
- IP addresses - <https://www.kaspersky.com/resource-center/definitions/what-is-an-ip-address>
 - Basically just names/ unique identifiers for things while endpoints are how to access services
- GCC - GNU Compiling Collections - to compile C and C++ langs
 - List of compilers: C (**gcc**), C++ (**g++**), Objective-C, Objective-C++, Java (**gcj**), Fortran (**gfortran**), Ada (**gnat**), Go (**gccgo**), OpenMP, Cilk Plus, and OpenAcc
 - GNU - GNU's not Unix - OS replacement for Unix
 - Unix - OS, can do multi tasking and from 1970s, closed source code
 - Linux - GNU plus Linux kernel, UNIX clone
 - Kernel -computer program facilitates interactions between hardware and software components
 - ◻ Does mem management, process management, device drivers
 - ◻ <https://www.redhat.com/en/topics/linux/what-is-the-linux-kernel>
- Linux distros

- Centos - Red Hat Enterprise Linux, yum, less freq package updates (more stable), maybe better for business (more secure, stable)
- Ubuntu- Debian, apt-get, has more latest software (more freq updates), more popular (more material)
- **Maven**
 - Type of software project management- does build, managing, doc
 - For java projects, good for downloading resources, for easily getting JAR files
 - Uses **Pom.xml** as core config profile, plugins, needed to tell project how to build
 - **Mvn package** to build
 - <https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html>
 - Ant - older, more flexible, java and non java, everything written from scratch (XML)
 - Different phases called 'target's
 - Gradle - built on Ant, Maven - based on domain specific language (Groovy/Kotlin)
 - Plugin heavy, has 'tasks', adv analysis and debugging, faster than Maven
- **.NET framework**
 - Platform for building software, languages used on it are c# and visual basic
 - Can run on windows (platform) and others OS (core), to create and run websites, services apps
 - <https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet-framework>
- **Nix**
 - <https://serokell.io/blog/what-is-nix>
 - Tool that has package manager, nix language, has reproducible builds, cross-distro/platform compatibility, lots of packages
 - Regular Linux filing makes it hard to reproduce well since cant see info on compiled files such as version control, configs, libraries used
 - Can fix versions using Docker, Snap, Flatpak but problematic
 - NixOS - linux distro uses nix package and config manager
 - Don't confuse with * nix system which is unix like OS
- **Puppet**
 - Pull model (as opp to Ansible push model) - works by client (Puppet agent, daemon) pulling from Puppet master which manages config for all nodes
 - Uses Ruby, also declarative (like kube yml)
 - Agents send facts to master, master sends back catalog that client needs to fix
 - <https://www.guru99.com/puppet-tutorial.html>
- **Qclapp /qclweb**
- **Schmeck**
- **Splunk - security**
 - For collecting machine data, processing and extracting relevant data from websites, apps, devices etc
 - Can work in real time, analyze patterns, send notif based on data
 - Works with Forwarder (collects data), Indexer (process data in real time), Search Head (how end users interact)
 - input, parse (divide into events), indexing (events sorted and indexed for storage), search (gain intell. and report)
- **SVN - Subversion**
 - Centralized version control system for source ccode, doc, files - like git
 - Atomic commits, full revision history, file locks, etc
 - Svn server (controls repo) and svn client (commit/update)
 - Difference from git - with svn only one central repo, git has local repos (commits) then that can go to master (push)