USGS
science for a changing world

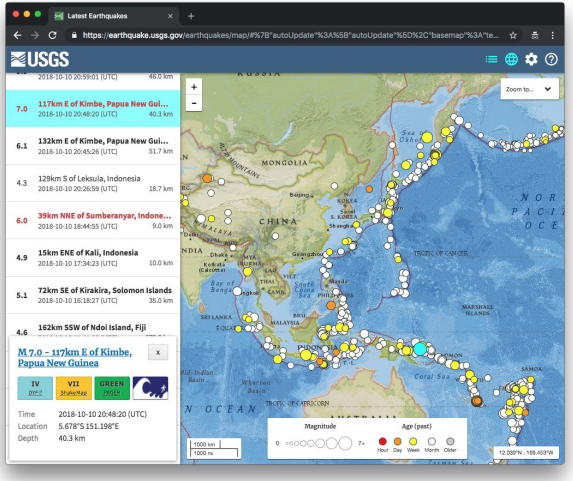
Docker Image Development

Best practices for creating Docker images

DVLP DNVR | Denver, CO | 2018-10-18

USGS
science for a changing world

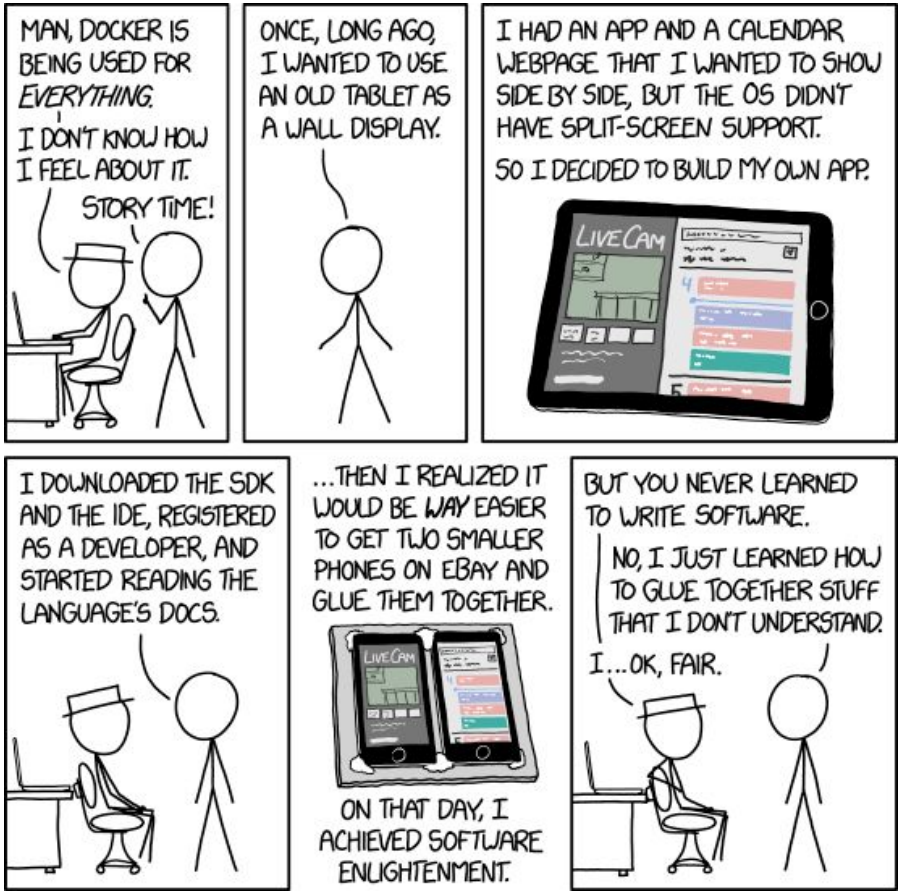
Eric Martinez
emartinez@usgs.gov
GitHub: @emartinez-usgs



United States Geological Survey
National Earthquake Information Center
https://earthquake.usgs.gov/

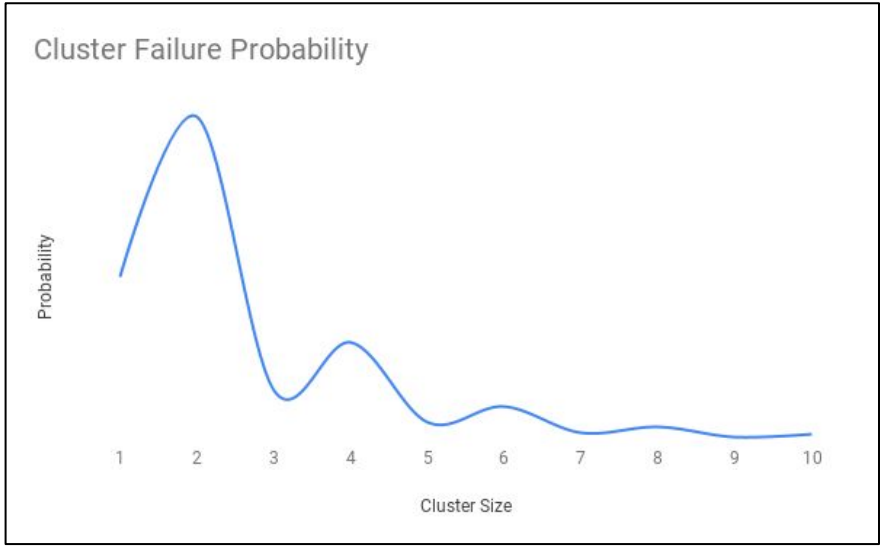
Scope / Outline

- Infrastructure
- **Image Development**
- Deployment Considerations



Infrastructure

- Servers
  - Managers/workers
  - Labels
- Image Registry
  - Trusted/public
  - Signing/scanning
- Automation
  - Build, test, deploy



Cluster Failure* Probability = $\binom{N}{K+1} \cdot P^{(K+1)}$

N: Number of nodes in the cluster
K: Cluster tolerance
P: Probability of any single node failing

*Raft concensus can not be reached. Existing replicas continue to function, but no action can be taken to change/correct current state.

Automation

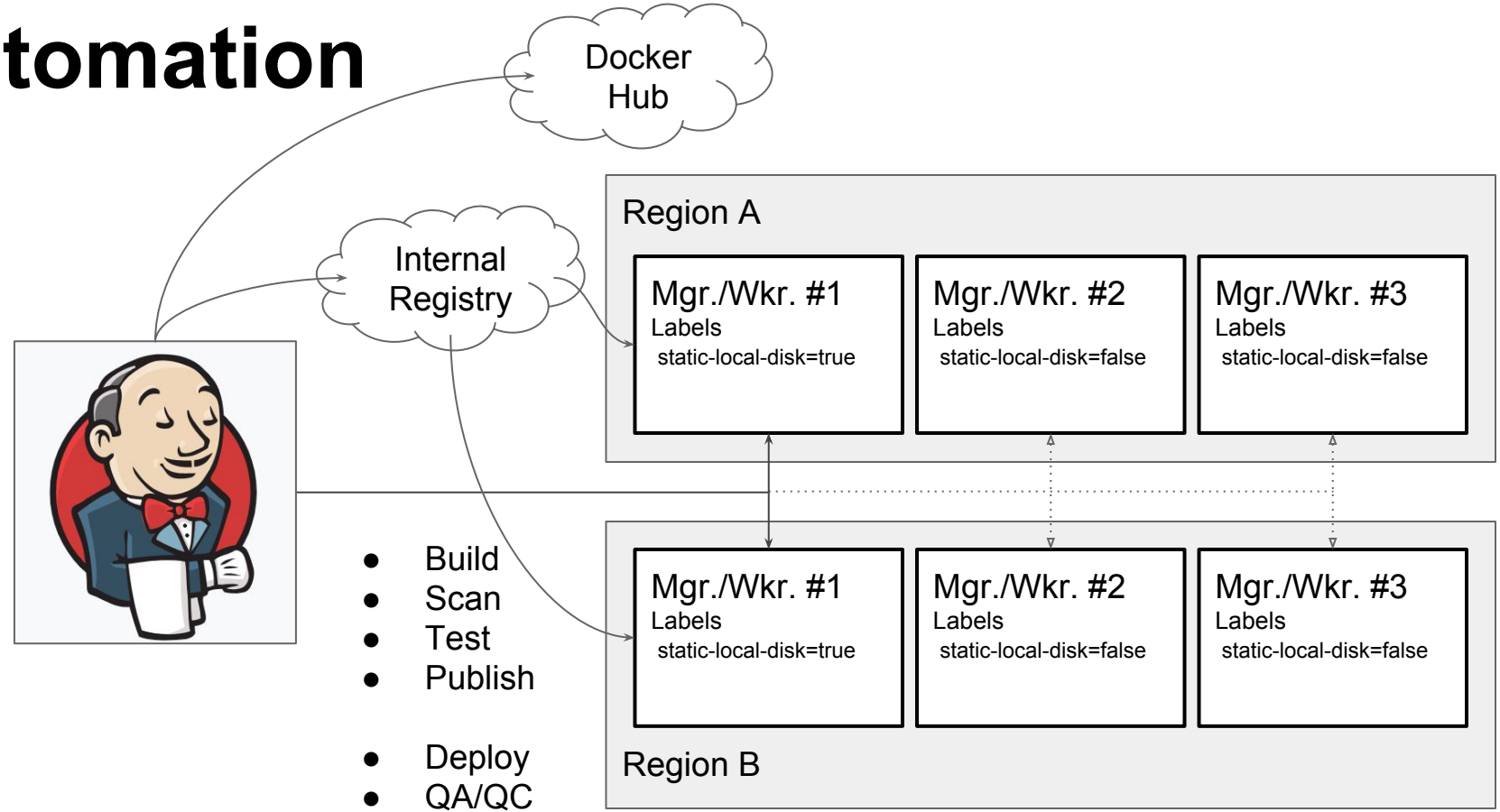

Docker Hub
Internal Registry
Region A
Mgr./Wkr. #1
Labels
static-local-disk=true
Mgr./Wkr. #2
Labels
static-local-disk=false
Mgr./Wkr. #3
Labels
static-local-disk=false
Mgr./Wkr. #1
Labels
static-local-disk=true
Mgr./Wkr. #2
Labels
static-local-disk=false
Mgr./Wkr. #3
Labels
static-local-disk=false
Region B
Build
Scan
Test
Publish
Deploy
QA/QC

Image Development

Dockerfile

https://docs.docker.com/engine/reference/builder/

- **ARG**
- FROM
- ENV
- LABEL
- RUN
- USER
- WORKDIR
- **COPY**
- EXPOSE
- **HEALTHCHECK**
- STOPSIGNAL
- **ENTRYPOINT / CMD**

ARG

In Dockerfile (provides a default)

```
ARG name=value
```

On command line (provides preferred value)

```
$ docker build --build-arg name=value …
```

Can be used before FROM to change base image.

Multi-Stage Builds

Example: Dockerfile

- Help keep distributed image size small
- Download install build tools in builder stage
- Copy artifacts out of builder stage into distributable image

COPY

In general, copies files/directories from build context to image file system.

- --from=stage
- --chown=user:group

```
COPY --from=buildenv \
    --chown=usgs-user:usgs-user \
    /earthquake-eventpages/dist/ \
    /usr/share/nginx/html/BASE_HREF/

COPY --from=buildenv \
    --chown=usgs-user:usgs-user \
    /earthquake-eventpages/metadata.json \
    /usr/share/nginx/html/BASE_HREF/metadata.json
```

HEALTHCHECK

Necessary to avoid downtime during upgrades.

- timeout
- interval
- start-period
- retries
- command

```
HEALTHCHECK \
    --interval=20s \
    --timeout=5s \
    --start-period=1m \
    --retries=2 \
    CMD \
    ${HEALTHCHECK_SCRIPT}
```

Healthchecks

```bash
#!/bin/bash -e

host=$(hostname -i || echo '127.0.0.1');
user="${POSTGRES_USER:-postgres}";
db="${POSTGRES_DB:-$user}";
export PGPASSWORD="${POSTGRES_PASSWORD:-}";

args=(
  --host "${host}"
  --username "${user}"
  --dbname "${db}"
  --quiet --no-align --tuples-only
);

select=$(echo 'SELECT 1' | psql "${args[@]}");
result=$?;

if [[ $result -eq 0 && $select -eq 1 ]]; then
  echo '[HEALTHCHECK] Database up and accepting connections.';
  exit 0;
fi

echo '[HEALTHCHECK] Database not healthy.';
exit 1;
```

```bash
#!/bin/bash -e

host=$(hostname -i || echo '127.0.0.1');
port="${PORT:-8000}";
mount_path="${MOUNT_PATH:-/ws/designmaps}";

args=(
  -s
  -o /dev/null
  -w '%{http_code}'
  -A 'Internal Healthcheck'
  "http://${host}:${port}${mount_path}/"
);

http_code=$(curl "${args[@]}");
result=$?;

if [[ $result -eq 0 && $http_code -eq 200 ]]; then
  echo '[HEALTHCHECK] Webserver up and accepting connections.';
  exit 0;
fi

echo '[HEALTHCHECK] Webserver not healthy.';
exit 1;
```

Another example: [healthcheck.sh]

Demo

https://github.com/emartinez-usgs/dvlp-dnvr-demo/tree/master/healthcheck

ENTRYPOINT / CMD

- Use *exec* form of ENTRYPOINT instruction to specify how the container starts
  - Include *required* arguments here
- Use exec form of CMD instruction to specify defaults for optional arguments
- Important to understand the difference

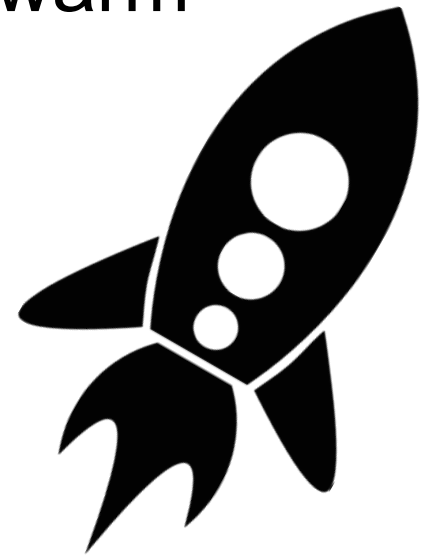
Configuration / Bootstrapping

Example: docker-entrypoint.sh

- Take action during container startup
  - Read configuration, environment, secrets
  - Write configuration, update deployment
- Manage signal processing
  - Important if main process does not handle natively

Deployment

- Deploy application as stack in swarm
- Use YML configuration file
- Use environment variables
- Use config/secret
- Service discovery



Example

```
$ env
  IMAGE_NAME=my/image:latest
  REGISTRY=dtr.internal.domain
  …

$ docker stack deploy \
    --prune \
    --with-registry-auth \
    --resolve-image always \
    -c deployment.yml \
    my-cool-app
```

YML File

```yaml
version: "3.5"
services:
  nginx:
    image: ${REGISTRY}/${IMAGE_NAME}
    deploy:
      replicas: 3
      placement:
        constraints:
          - node.labels.some-label == value
      restart_policy:
        condition: any
        delay: 5s
        max_attempts: 2
        window: 60s
      update_config:
        parallelism: 1
        delay: 5s
        failure_action: rollback
        order: start-first
    environment:
      - DB_HOST
      - POSTGRES_USER=${DB_USER}
      - PGPASSWORD=${DB_PASSWORD}
    ports:
      - "80"
    configs:
      - source: update-config
        target: /update-config.sh
      - source: remove-config
        target: /remove-config.sh
configs:
  update-config:
    file: ./update-config.sh
  remove-config:
    file: ./remove-config.sh
```

Service Discovery

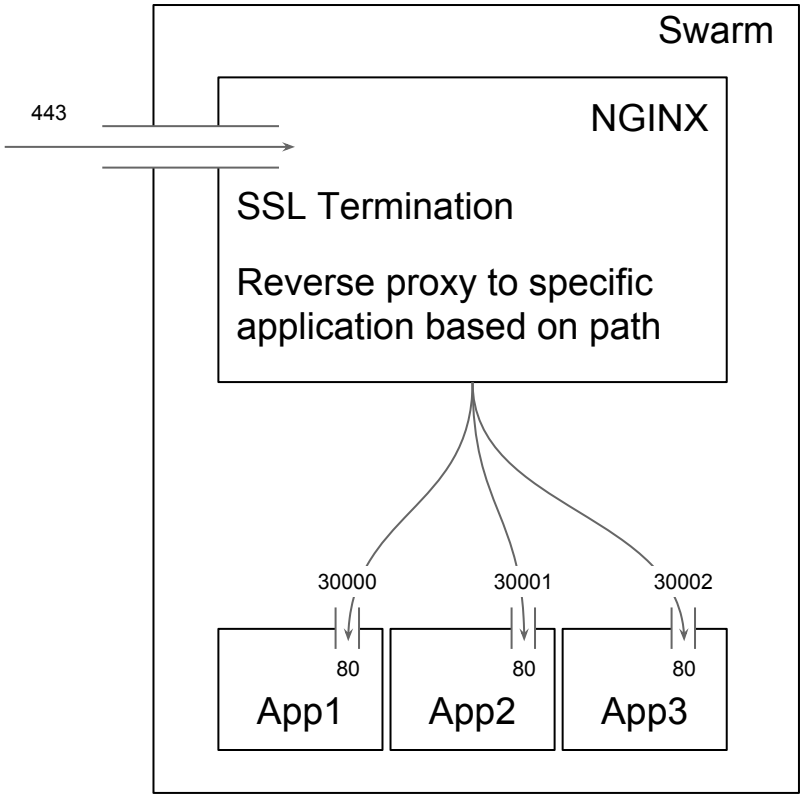
Discover port exposed port for application

```
$ docker inspect --format \
  '{{index .Endpoint.Ports 0).PublishedPort}}' \
  my-cool-app
```

Reconfigure NGINX

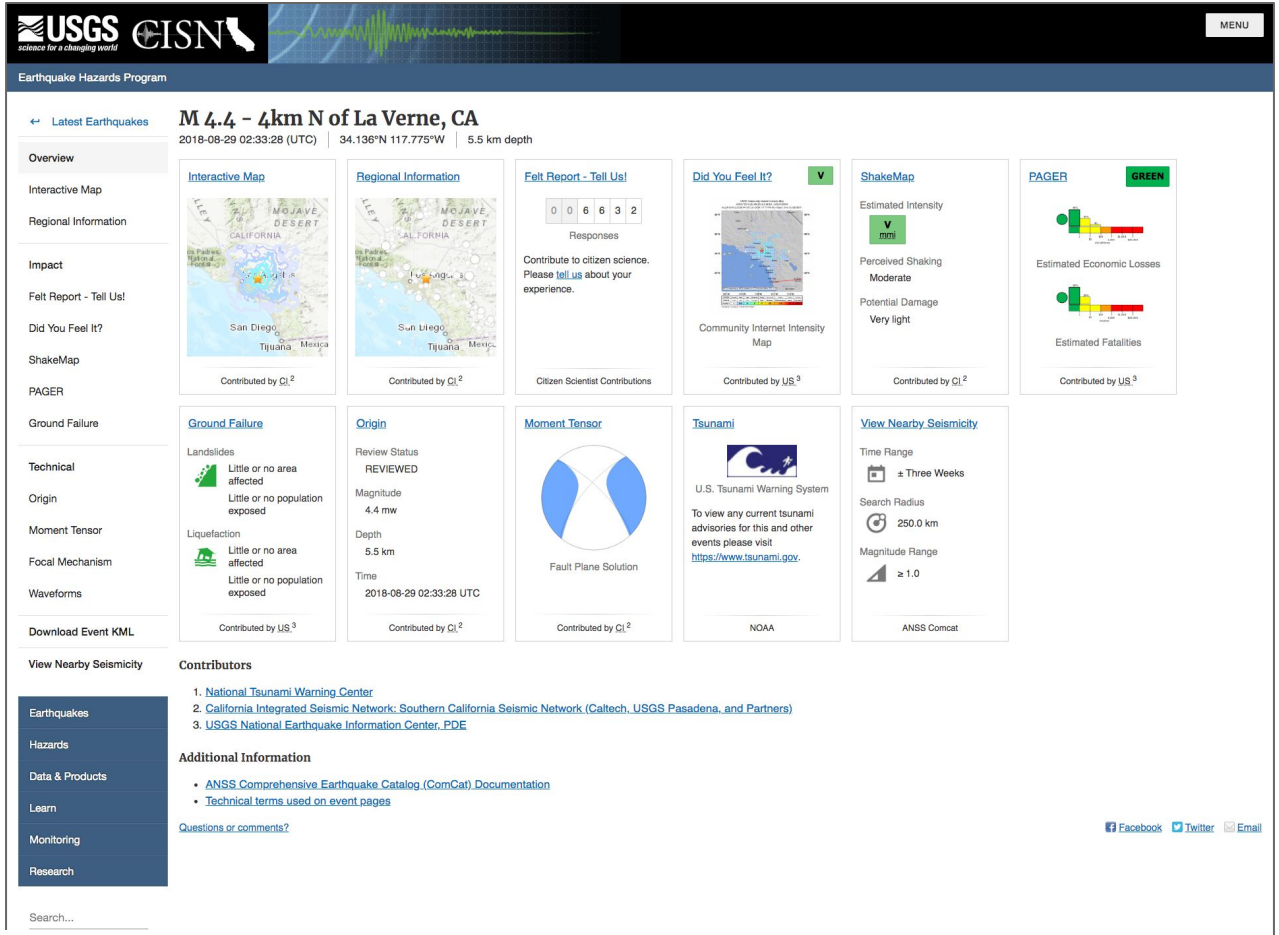
```
upstream my-cool-app_nginx {
  server worker1:30000;
  server worker2:30000;
  server worker3:30000;
}

location /some/path/ {
  proxy_pass http://my-cool-app_nginx;
  proxy_set_header Host my.website.example;
  proxy_set_header X-Client-IP $remote_addr;
}
```



Questions?

Demo



Also, all this stuff actually works...

Thank You

So long, and thanks for all the fish.