Table of Contents

FIMpy - Python File Integrity Monitoring App

Source: https://oneillal.github.io/FIMpy

Exported from GitHub using GitBook

Monthly Journal 1

The Idea

After toying with a few initial ideas, I talked to our in house security team because I wanted a real world security related problem for my project. After a quick chat, a good idea came about. A file integrity monitoring solution, or FIM. I had to do a little research. Turns out, there are many tools already that do something similar. Tripwire, Aide, Samhain to name a few. Almost all are built in C derivatives for obvious performance reasons. We can't use open-source on a PHI platform so a home-grown solution might be a good fir. The plan is to write an application which addresses our specific uses cases and requirements (which are mainly cloud driven) and develop with a modern programing language and using existing PAAS, libraries and services as much as possible.

RWE

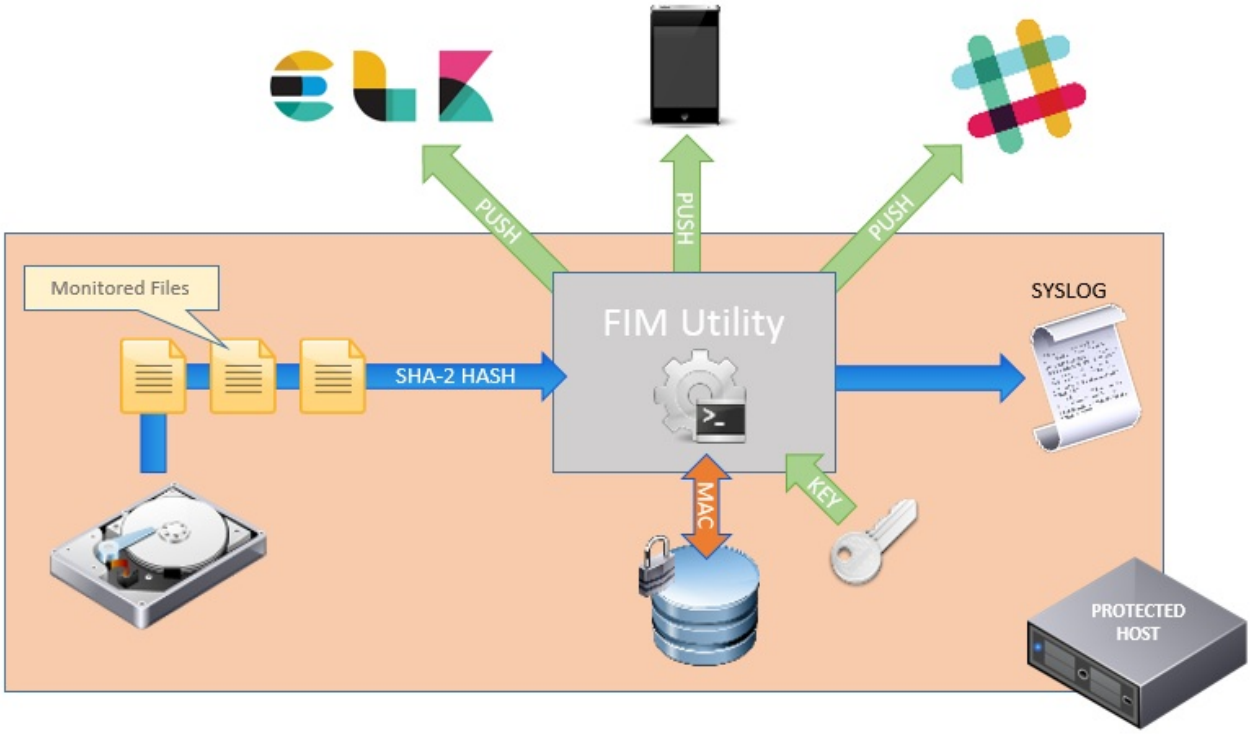
The treat of ransom-ware is very much to the forefront of Watson Platform for Health security concerns. Patient Health Information (PHI) is highly valued by cyber criminals and our in-house Ops and Security team are engaged in a continuous cycle of security evaluation and improvement. The threat of malware inside the firewall is almost of grave concern and the potential for PHI loss if very real. We recognised the need for an IDS solution that can detect malware and Trojans and also detect and prevent a ransom-ware attack e.g. CryptoLocker [1] which typically involves encryption of victim's files.

 # [1] https://en.wikipedia.org/wiki/CryptoLocker

Design & Requirements

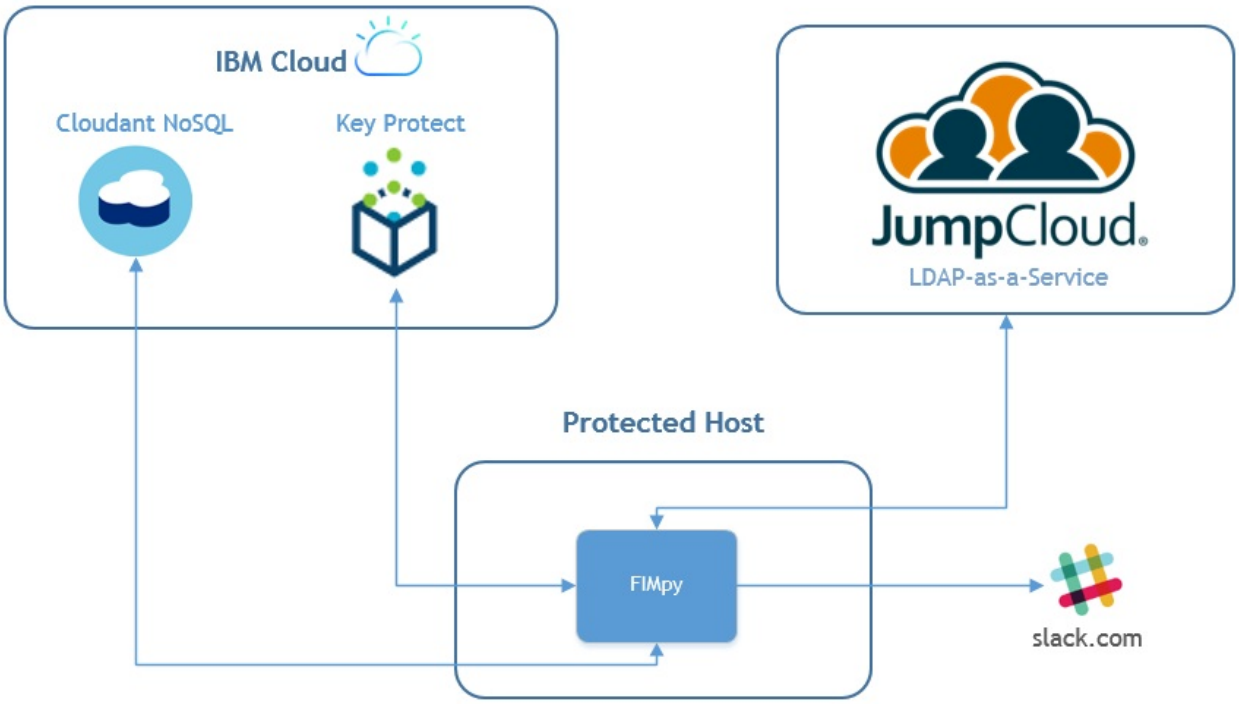
I've called the application FIMpy! Originally I called it pyFIM only to google and find a million unrelated things called pyfim. Oh. And here is FIMpy...!

The initial design from my proposal was for a standalone FIM utility but it doesn't work well in a Cloud environment. I want to take advantage of services and develop a micro-service driven solution. That doesn't make sense if everything is local. What it you want to monitor many different hosts? I need to spend more time thinking about the solution architecture.



Update: I'm starting to look at a client-server architecture using a master and slave approach. Any number of slaves deployed on a nodes to monitor files and send back integrity information to the master which verifies integrity by comparing information with that stored in a secure database. That will of course be a lot more work. I need to develop the client first and then look at the server. It needs to be micro-service driven so Flask is perfect. So all communication can be via REST.

New design using services...



NFR will be very important:

- Performance - Python/NoSQL may not be best. Cython [1] can help with speed. We'll see how Cloudant performs with queries.
- Security - goes without saying. SSL encryption, secure authentication.
- Auditing - will need to be auditable.

[1] https://en.wikipedia.org/wiki/Cython

Monthly Journal 2

Python

I have decided to use Python to develop my solution due to the flexibility of the language and the speed at which results can be achieved. Python also offers a huge array of community supported libraries and functions that I hope to be able to take advantage of. Python also has numerous web frameworks. I am going to utilise the Flask framework for my application using a micro-services approach. Flask is also supported by Bluemix which is where I plan to host my Python application. I have very little practical experience with Python.

Update: Python rules!

HMAC

I must admit it took a while to get my head around HMAC. I watched a bunch of YouTube videos and even then it wasn't making a whole lot of sense. I mean it was easy to generate hmac's and hashes in my Python code using the hashlib [1] and hmac [2] libraries. What didn't make sense to me was why I didn't always getting the same hmac for the same file? Turns out in the end it was an initialisation error but it took a while to figure it out. I wasn't reinitialising the hmac within a loop and so I was calculating a running HMAC for all the files I was monitoring.

With hmac and hashlib, you use the update() method because it's advisable to read a certain number of blocks at a time and not read the whole file on one go. So you need to create a new HMAC for each and every file.

```
for file in glob.glob(os.path.join(path, '*')):
    if os.path.isfile(file):
        f = open(file, 'rb')
        try:
            sha256 = hashlib.sha256()
            digest = hmac.new('key', '', hashlib.sha256)
            while True:
                block = f.read(BUF_SIZE)
                if not block:
                    break
                digest.update(block)
                sha256.update(block)
        finally:
            f.close()
```

[1] https://docs.python.org/2/library/hashlib.html

[2] https://docs.python.org/2/library/hmac.html

Authentication

So turns out that AppID is not what I need. There is an third party IAM service called PAssport but it doesn't fall under the free IBMer Bluemix plan. It has a 14 day trial but that obviously won't work. I really need authentication and I really don't want to write reams of code to implement it.

Looking outside Bluemix, I came across JumpCloud LDAP-as-a-Service. They offers a free plan for max of 5 users so I'll give it a try.

Slack

Slack is everywhere these days. It's pretty much the defacto collaboration platform now so I plan to send FIMpy alerts to Slack. Should be easy enough to use a web hook at a minimum to fire not a slack channel. I've done this before in bash scripts using curl and I'm assuming (hoping) there are Python libraries for this.

Update: Found this Python interface for the Slack: https://github.com/os/slacker Looks perfect for my needs. Looks like I need a Slack developer API key. Need to read up on this.

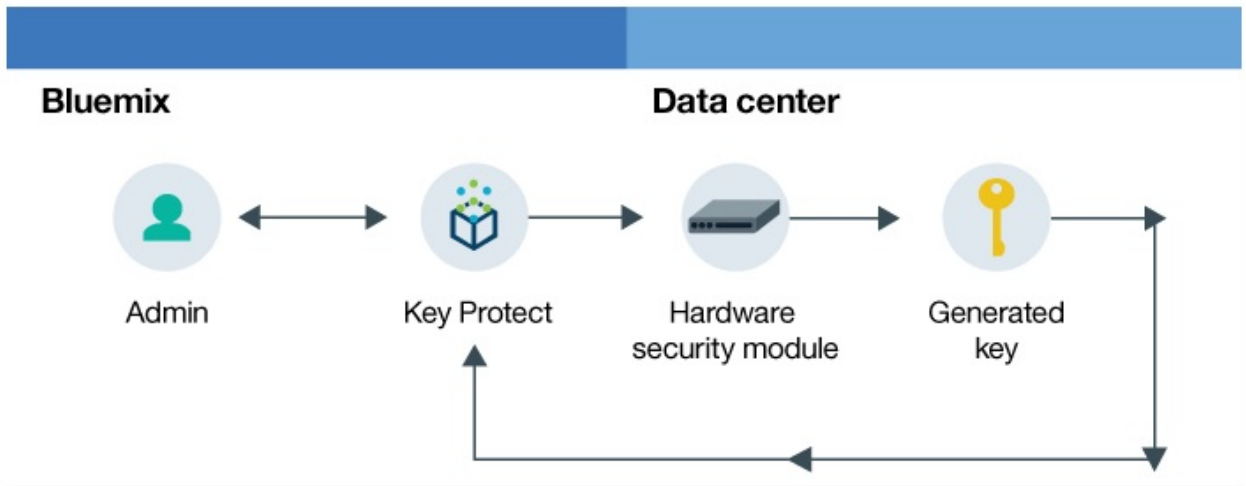
Bluemix

Bluemix is an excellent platform to learning new technologies. I've play with many runtimes and services over the last year. It supports Flask web frameworks and has a number of different data storage solutions. Initially I planned to run a native Python app using Flask but that got superceeded by Docker and then again by Kubernetes.

I have plenty of experience with DB2 so I plan to use Cloudant [1] instead for a data solution. I want to get more hands on with NoSQL. It also has very good Python support and there is great tutorial here.

I hope to use AppID for authentication but I am very familiar with it and it looks like it might be more of a IDP/SSO type service.

There is also an interesting new service called Key Protect [2] that I plan to take a look at. It is a secure key management service which is exactly what I need for generating and validating HMAC's.



[1] https://console.bluemix.net/docs/services/Cloudant/cloudant.html#overview

[2] https://console.bluemix.net/docs/services/keymgmt/keyprotect_about.html#about-key-protect

Containers

Spent most of the time this week trying to containerise the application. I'm heavily invested in DevOps in my day job so I've played a bit Docker and regularly attend meetups. Before the app can be deploy on the Watson platform, it will need to run in a container. Time spent getting everything working could probably have been spent on other features but once I start on something, I can't leave it alone.

```
$ docker build -t alanoneill/fimpy:latest .
```

```
FROM ubuntu:zesty
MAINTAINER "alan.oneill75@gmail.com"
RUN apt-get update -y
RUN apt-get install -y python-pip python-dev build-essential libldap2-dev libsasl2-dev libssl-dev
COPY . /app
WORKDIR /app
RUN pip install -r requirements.txt
EXPOSE 5000
ENV SLACKTOKEN=
#ENTRYPOINT ["python"]
#CMD ["app.py"]
CMD ["python","-u","app.py"]
```

Kubernetes

I'm a bit obsessed with Kubernetes at the moment. I first became aware of it watching a Google Dev conference on YouTube a couple of years ago. Bluemix (soon to be known as IBM Cloud) offers a free one node cluster so of course I could stop with just running FIMpy in a container, I had to deploy it within a Kube cluster!

In order to get the deployment to work on a Bluemix cluster, I needed to publish my image to dockerhub. I could of also created a private Bluemix registry but I already had a dockerhub account from yester-year. Took a little time to figure out the networking with Kubernetes. I couldn't get the internal exposed 5000 port exposed to an external port. With a little help from a collegue, turns out the issue was with the `apiVersion`. Once that was fixed it worked a treat. Next task is to get the SSL working within the cluster.

Here's the deployment yaml.

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: pyfim-app
  labels:
    app: pyfim-app
spec:
  selector:
    matchLabels:
      app: pyfim-app
  template:
    metadata:
      labels:
        app: pyfim-app
    spec:
      containers:
      - name: pyfim-app
        image: alanoneill/pyfim
        ports:
        - containerPort: 5000
```

And the service yaml.

```
apiVersion: v1
kind: Service
metadata:
  name: pyfim-service
  labels:
    name: pyfim-app
spec:
  type: NodePort
  ports:
    - port: 5000
      targetPort: 5000
      protocol: TCP
      name: pyfim-service
  selector:
    app: pyfim-app
```

Testing

POST /api/fimpy - Write monitored file info to db

```
$ curl -k -X POST -u admin:password --url https://127.0.0.1:5000/api/fimpy
Records added to db
```

GET /api/fimpy - Read monitored file info from db

```
$ curl -k -u admin:password --url https://127.0.0.1:5000/api/fimpy
[
  "/app/test/11kfile",
  "/app/test/sniff.py",
  "/app/test/jumpcloud_test_utility.sh",
  "/app/test/10kfile"
]
```

LDAP Authentication Failure

```
$ curl -v -k -u admin:badpw --url https://127.0.0.1:5000/api/fimpy/scan
*   Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to 127.0.0.1 (127.0.0.1) port 5000 (#0)
* ALPN, offering http/1.1
* Cipher selection: ALL:!EXPORT:!EXPORT40:!EXPORT56:!aNULL:!LOW:!RC4:@STRENGTH
* successfully set certificate verify locations:
*   CAfile: /etc/ssl/certs/ca-certificates.crt
  CApath: /etc/ssl/certs
* TLSv1.2 (OUT), TLS header, Certificate Status (22):
* TLSv1.2 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS change cipher, Client hello (1):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS change cipher, Client hello (1):
* TLSv1.2 (IN), TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / ECDHE-RSA-AES256-GCM-SHA384
* ALPN, server did not agree to a protocol
* Server certificate:
*  subject: C=IE; ST=DUB; L=Dublin; O=IBM; OU=Watson; CN=xubuntu
*  start date: Nov  2 19:44:48 2017 GMT
*  expire date: Nov  2 19:44:48 2018 GMT
*  issuer: C=IE; ST=DUB; L=Dublin; O=IBM; OU=Watson; CN=xubuntu
*  SSL certificate verify result: self signed certificate (18), continuing anyway.
* Server auth using Basic with user 'admin'
> GET /api/fimpy/scan HTTP/1.1
> Host: 127.0.0.1:5000
> Authorization: Basic YWRtaW46YmFkcHc=
> User-Agent: curl/7.52.1
> Accept: */*
>
* HTTP 1.0, assume close after body
< HTTP/1.0 401 UNAUTHORIZED
* Authentication problem. Ignoring this.
< WWW-Authenticate: Basic realm="Login Required"
< Content-Type: text/html; charset=utf-8
< Content-Length: 20
< Server: Werkzeug/0.12.2 Python/2.7.13
< Date: Sun, 05 Nov 2017 19:18:25 GMT
<
```

```
* Curl_http_done: called premature == 0
* Closing connection 0
* TLSv1.2 (OUT), TLS alert, Client hello (1):
Authenticaton failed
```

References

Bluemix Getting Started Guide

https://console.bluemix.net/docs/cli/reference/bluemix_cli/get_started.html#getting-started

Python Flask SSL

http://bobthegnome.blogspot.co.uk/2007/08/making-ssl-connection-in-python.html http://flask.pocoo.org/snippets/111/

Python LDAP

https://www.packtpub.com/books/content/python-ldap-applications-part-1-installing-and-configuring-python-ldap-library-and-bin

Python Flask Basic Auth Sample Code

http://flask.pocoo.org/snippets/8/

LDAP-as-a-Service

https://jumpcloud.com/daas-product/ldap-as-a-service

Python Slack Wrapper

https://github.com/os/slacker

Docker User Guide

https://docs.docker.com/engine/installation/linux/docker-ce/ubuntu/

DockerHub - Publishing

https://ropenscilabs.github.io/r-docker-tutorial/04-Dockerhub.html

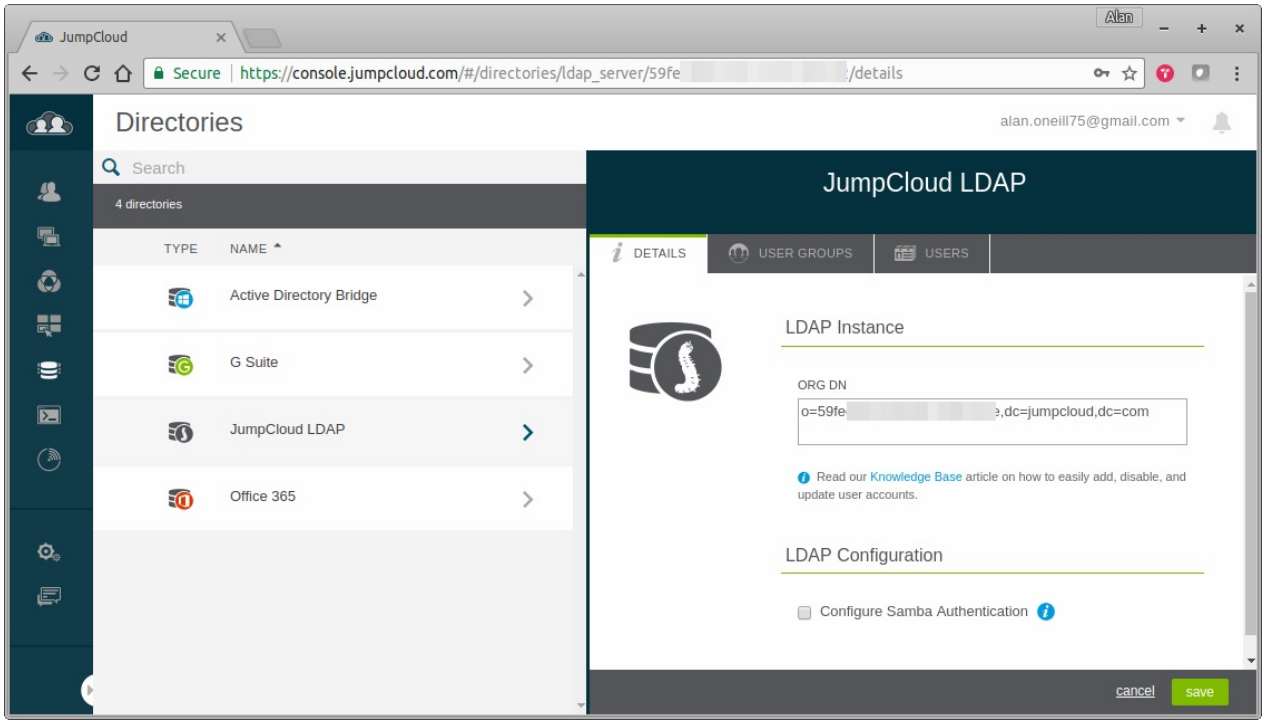
Kubernetes Docs

https://kubernetes.io/docs/home/

FIMpy Logo

https://www.easyicon.net/language.en/541685-face_monkey_icon.html (Free for commerical use)

JumpCloud LDAP-as-a-Service



Docker Reference

Building the FIMpy Image

```
~/dev/project/FIMpy/app(master*) » docker build -t alanoneill/fimpy:latest .
Sending build context to Docker daemon  71.17kB
Step 1/10 : FROM ubuntu:zesty
 ---> 5e9fde03a0de
Step 2/10 : MAINTAINER "alan.oneill75@gmail.com"
 ---> Using cache
 ---> 5914a7f13422
Step 3/10 : RUN apt-get update -y
 ---> Using cache
 ---> e5f757b323dc
Step 4/10 : RUN apt-get install -y python-pip python-dev build-essential libldap2-dev libsasl2-dev libssl-dev
 ---> Using cache
 ---> 814066000105
Step 5/10 : COPY . /app
 ---> Using cache
 ---> 56cdbebdee06
Step 6/10 : WORKDIR /app
 ---> Using cache
 ---> 8ff32a4630b2
Step 7/10 : RUN pip install -r requirements.txt
 ---> Using cache
 ---> 0c9d470d64cf
Step 8/10 : EXPOSE 5000
 ---> Using cache
 ---> 725ca2626d92
Step 9/10 : ENV SLACKTOKEN
 ---> Using cache
 ---> 0e559213ef1e
Step 10/10 : CMD python -u main.py
 ---> Using cache
 ---> a052e2fe335b
Successfully built a052e2fe335b
Successfully tagged alanoneill/fimpy:latest
```

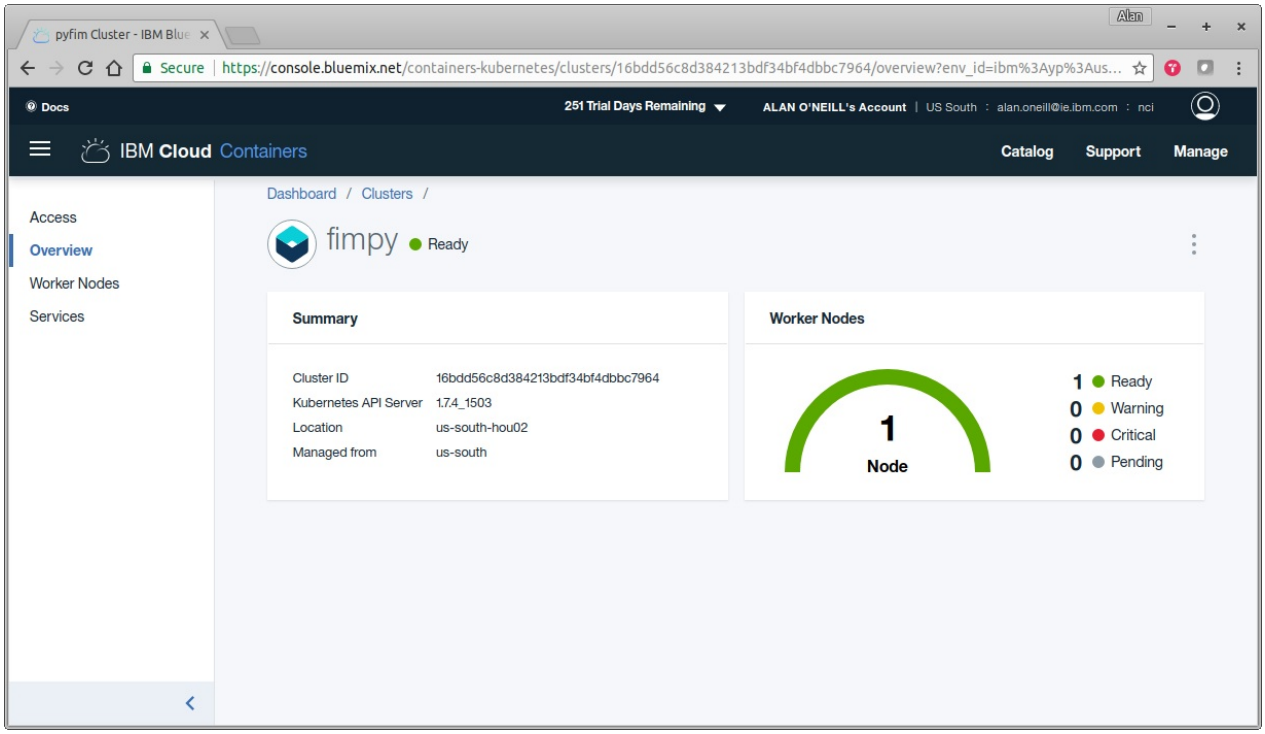
Starting a FIMpy Container

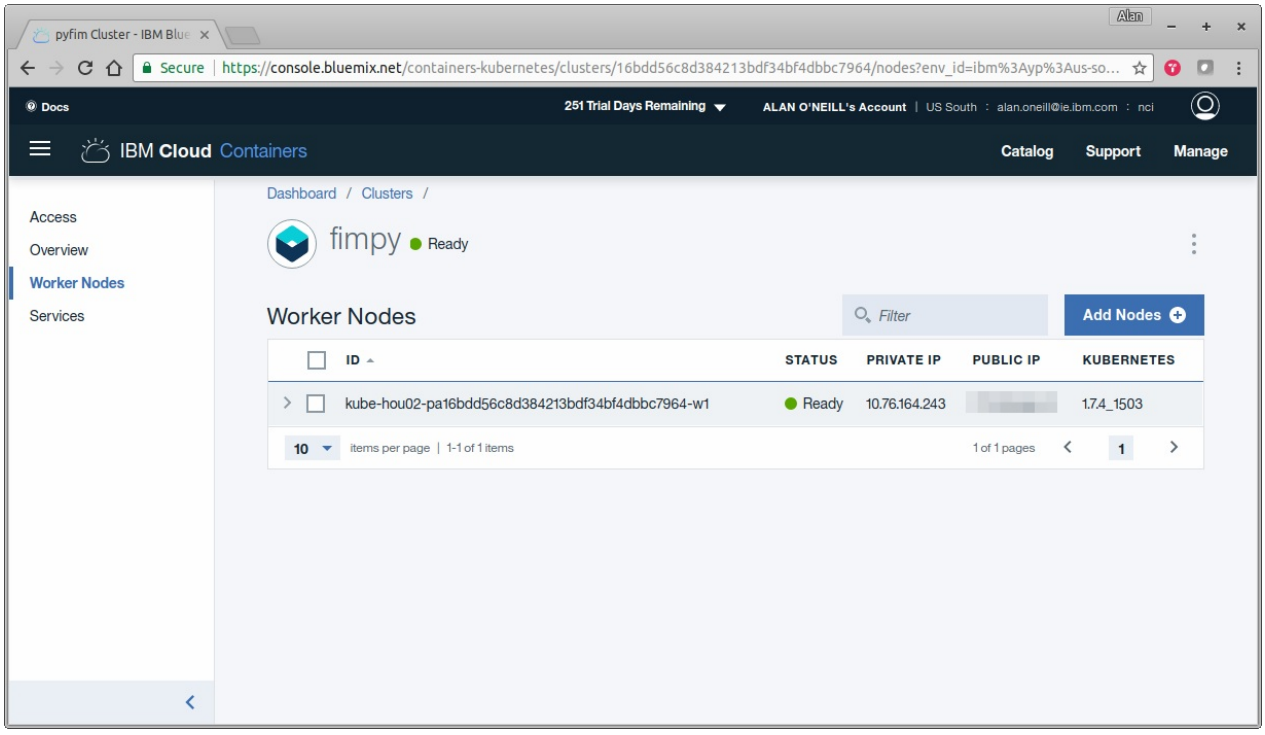
```
~/dev/project/FIMpy/app(master*) » docker run -d -p 5000:5000 --name fimpy alanoneill/fimpy
21a02c2de547b0957f1ac1a9ac0cc69b983ed8b29fa4f1a5a2b2a0c0ca368158
```

Push to DockerHub

```
~/dev/project/FIMpy/app(master*) » docker push alanoneill/fimpy
The push refers to a repository [docker.io/alanoneill/fimpy]
0bd94378fefa: Mounted from alanoneill/pyfim
5ac2b699e6fc: Mounted from alanoneill/pyfim
043f86434fa3: Mounted from alanoneill/pyfim
930be9908569: Mounted from alanoneill/pyfim
a5bfda07c31b: Mounted from alanoneill/pyfim
4cb653496843: Mounted from alanoneill/pyfim
69da000be6ed: Mounted from alanoneill/pyfim
3eaf555ee1db: Mounted from alanoneill/pyfim
71b5d87d106c: Mounted from alanoneill/pyfim
latest: digest: sha256:abde77d42f93270b12836181baae8615298229bc4a9aa2359997b70a0451bd06 size: 2203
```

Kubernetes Reference





Create Service

```
~/dev/project/FIMpy(master*) » kubectl create -f kube-service.yaml
service "fimpy-service" created
```

Create Deployment

```
~/dev/project/FIMpy(master*) » kubectl create -f kube-deployment.yaml
deployment "fimpy-app" created
```

Get Service

```
~/dev/project/FIMpy(master*) » kubectl get service -o wide

                        oneillal@xubuntu
NAME            TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE       SELECTOR
fimpy-service   NodePort    172.21.13.63   <none>        5000:31342/TCP   2m        app=fimpy-app
kubernetes      ClusterIP   172.21.0.1     <none>        443/TCP          10d       <none>
```

Get Pod

```
~/dev/project/FIMpy(master*) » kubectl get pods -o wide
NAME                          READY     STATUS    RESTARTS   AGE       IP              NODE
fimpy-app-3774997280-glctd   1/1       Running   0          2m        172.30.91.215   10.76.164.243
```

Logs

```
~/dev/project/FIMpy(master*) » kubectl logs -f fimpy-app-3774997280-glctd
Found local VCAP_SERVICES
fimpy-app-3774997280-glctd
 * Running on https://0.0.0.0:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 184-335-951
```

Deployment Shell Access

```
~/dev/project/FIMpy(master*) » kubectl exec -i -t fimpy-app-3774997280-glctd bash

root@fimpy-app-3774997280-glctd:/app# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 18:36 ?        00:00:00 /pause
root         5     0  0 18:36 ?        00:00:00 python -u main.py
root        15     5  1 18:36 ?        00:00:15 /usr/bin/python main.py
root        23     0  0 19:00 pts/0    00:00:00 bash
root        32    23  0 19:00 pts/0    00:00:00 ps -ef
```

GitBook Reference

Build and deploy script to publish FIMpy GitBook

```
#!/bin/bash
gitbook build
cd _book
git init
git remote add upstream git@github.com:oneillal/FIMpy.git
git fetch upstream
git reset upstream/gh-pages
touch .
rev=$(git rev-parse --short HEAD)
echo $rev
git add -A .
git commit -a -m "Rebuild gitbook pages at ${rev}"
git push -q upstream HEAD:gh-pages
```