

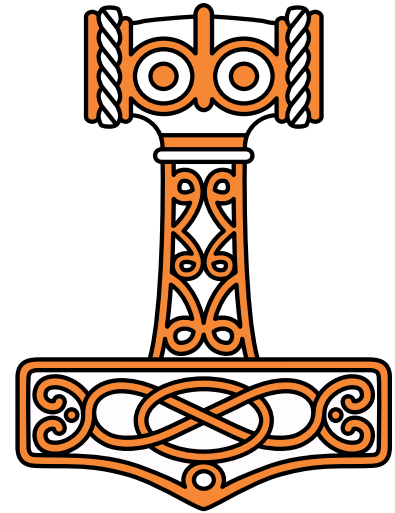
DYALOG

Olhão 2022

Simplifying Secure, Scalable Web Services

Session D12

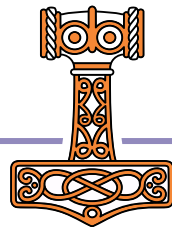
Brian Becker



Workshop Goals

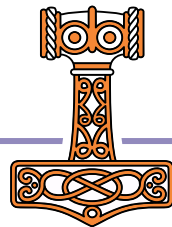
Give a quick introduction to:

- Jarvis – Dyalog's Web Service Framework – to expose APL functions as services
- Docker: to create lightweight Virtual Machines known as "Containers"
- Docker Compose: to launch and manage multiple inter-connected containers
- Amazon Web Services "Elastic Container Service": to allow Docker Compose to launch containers directly to the cloud (so-called "serverless" deployment)
- How to scale the system by running multiple copies of selected services
- How to assign your own domain name and a certificate to your service



Sunday Recap

- Workshop SA2 – Building Web Services with Jarvis
 - Jarvis – framework for REST and JSON-based web services
 - Important configuration settings, debugging, maintaining state
 - Introduced (barely) 3 versions of a "phonebook" application web service
- Workshop SP2 – Deploying Services
 - Started with the Jarvis phonebook application web service
 - By the end of the workshop ~70% of the participants had a load-balanced, scaled service running on AWS.
 - Demonstrated running it securely using HTTPS

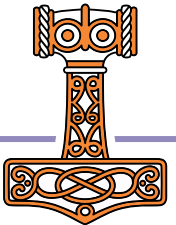
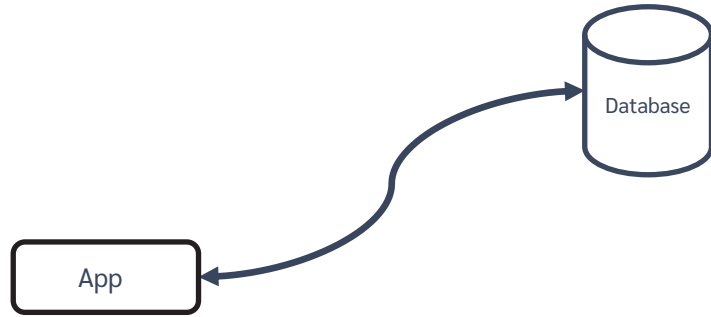


The "Plan"

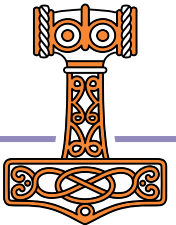
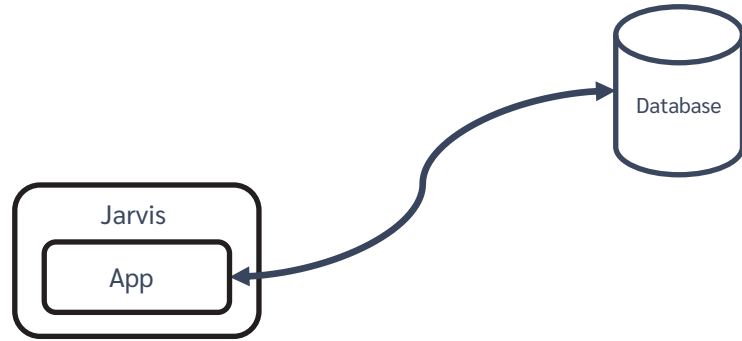
- Start with an application
- Use Jarvis to turn it into a web service
- Run the service in a Docker container
- "Split" the service into 2 tiers we termed "front end" and "back end" Each running in its own container.
Use Docker compose to define the containers that make up the service.
- Run the new service in the cloud (on AWS)
- Scale the service up by running multiple copies of the front end
- Add a load balancer to distribute requests amongst the front ends
- Secure the service by running HTTPS (requires a domain name and a certificate)



In the beginning, there was an Application...

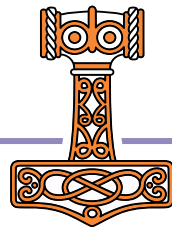


Run the app as a service



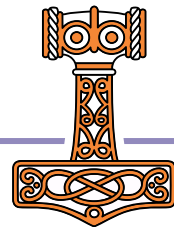
Jarvis - JSON and REST Services

- Web service framework to build REST or JSON-based services
- Using the JSON paradigm, APL functions are the "endpoints" of the service
 - The request data (payload) is passed to your function as an APL array
 - Your function returns an APL array
 - Jarvis handles all the conversion between JSON to APL and back again

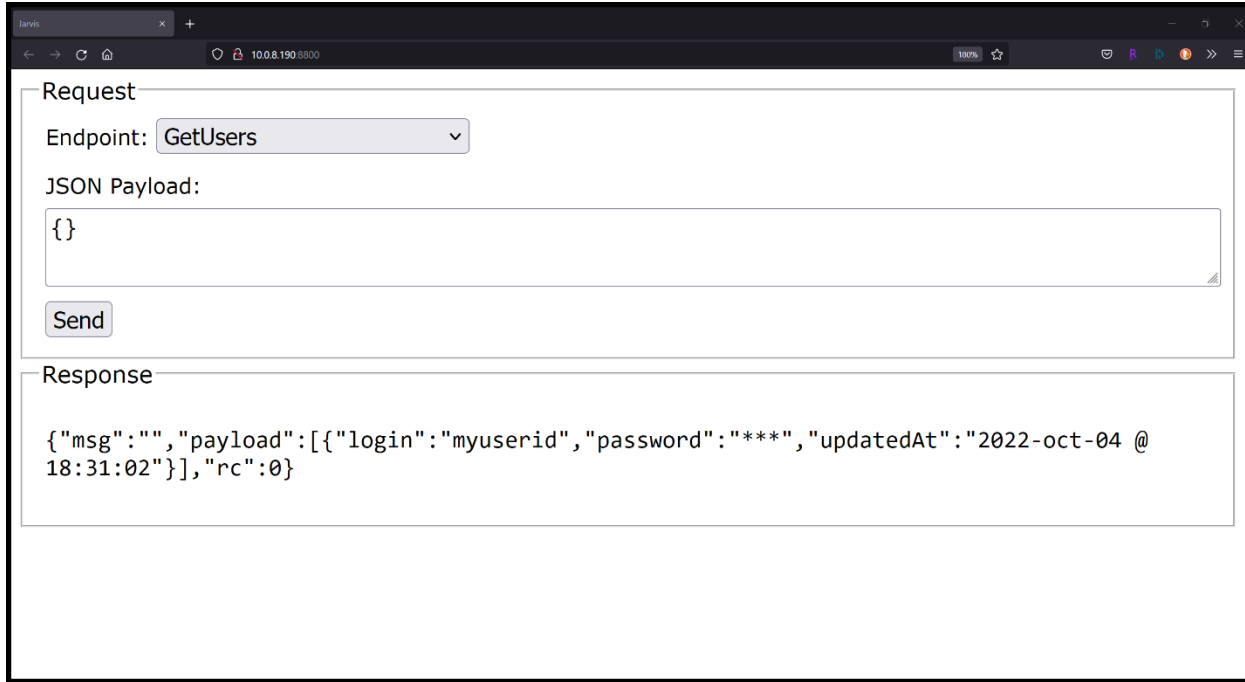


Jarvis in Action

```
]load [SA2]/Jarvis
#.Jarvis
  reshape<->ρ/
  hi<={'Hello ',ω.name}
  j<Jarvis.New 8888 #
  j.Start
  j.Stop
```



The app as a service



Request

Endpoint:

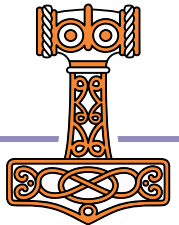
JSON Payload:

```
{}
```

Response

```
{"msg":"","payload":[{"login":"myuserid","password":"***","updatedAt":"2022-oct-04 @ 18:31:02"}],"rc":0}
```

The "GetUsers" endpoint being executed from a browser using a JavaScript client in Jarvis' built-in test web page.



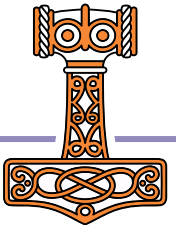
Any client, anywhere

- Any platform capable of issuing HTTP requests (basically everything) can interact with your web service.
- They need know nothing about APL nor that APL is running the web service

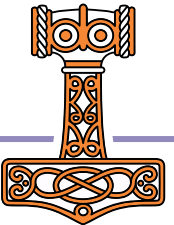
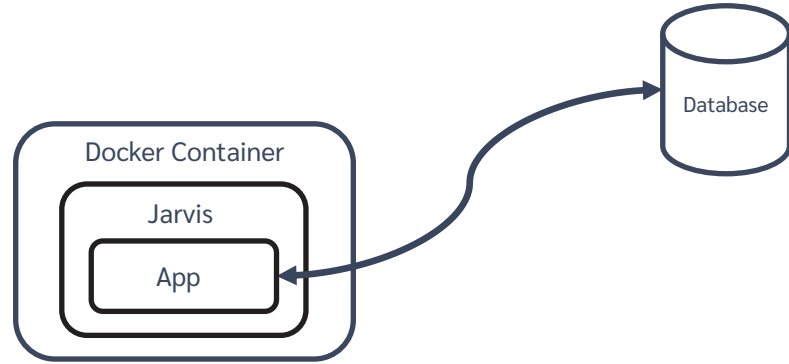
Administrator: Windows PowerShell

```
PS C:\Users\brian> curl -H "content-type: application/json" -d "{}" -X POST localhost:8800/GetUsers  
{"msg":"","payload":[{"login":"myuserid","password":"***","updatedAt":"2022-oct-04 @ 18:31:02"}],"rc":0}
```

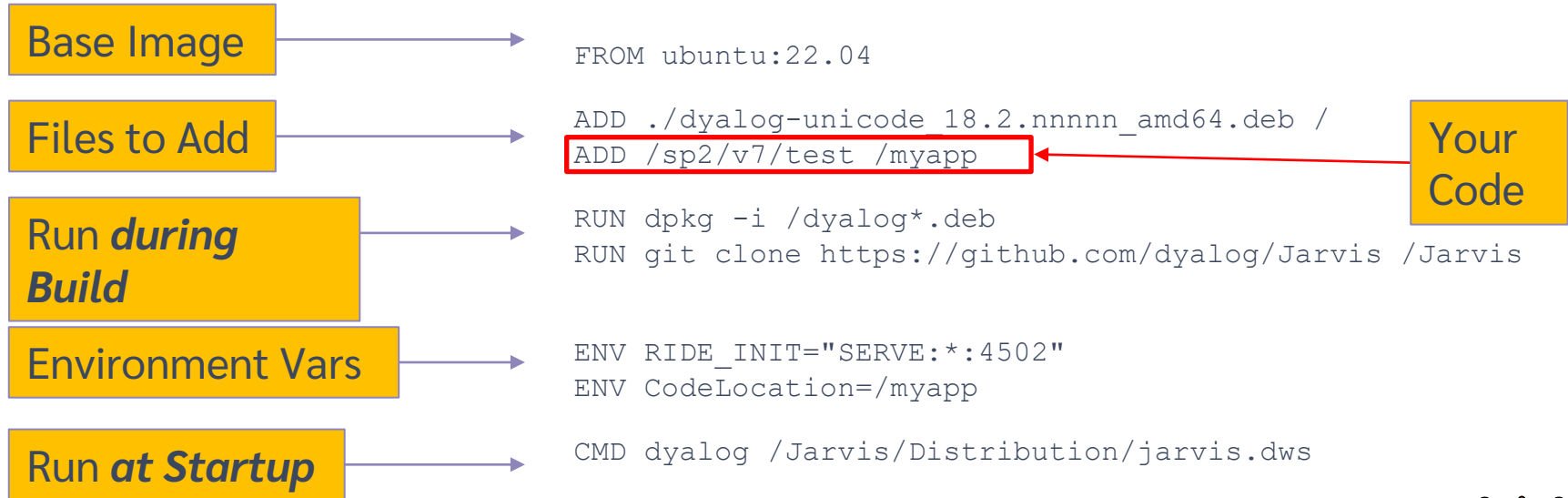
```
(HttpCommand.Do 'post' 'localhost:8800/GetUsers' ([NS'']) ('content-type' 'application/json')).Data  
{"msg":"","payload":[{"login":"myuserid","password":"***","updatedAt":"2022-oct-04 @ 18:31:02"}],"rc":0}
```



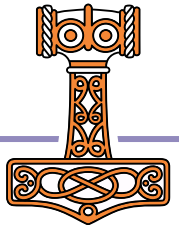
Run it in a container



A "Dockerfile" describes a Docker Image

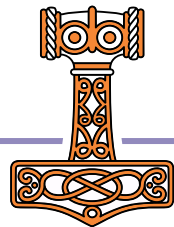


This "Dockerfile" completely describes a machine which will run "myapp".



Public Docker Containers

- ◆ DockerHub is to Docker as GitHub is to Git
 - ◆ A public repository of container images
 - ◆ Free unlimited public images plus one private image
 - ◆ Dyalog has several public containers available on DockerHub
 - ◆ `dyalog/dyalog` – Just Dyalog APL
 - ◆ `dyalog/miserver` – Dyalog APL + MiServer
 - ◆ `dyalog/jupyter` – Dyalog APL + Jupyter Notebook framework
 - ◆ `dyalog/jarvis` – Dyalog APL + Jarvis



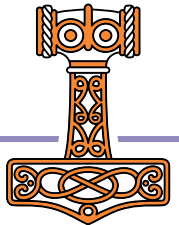
Running the App in a Container

```
PS C:\Users\brian> docker run -p 8080:8080 -v c:\sa2\v3:/app -e JarvisConfig=./JarvisConfig.json5 dyalog/jarvis

Dyalog

https://www.dyalog.com

Dyalog APL/S-64 Version 18.2.45405
Serial number: UNREGISTERED - not for commercial use
+-----+
| Dyalog is free for non-commercial use but is not free software. |
| A basic licence can be used for experiments and proof of       |
| concept until the point in time that it is of value.          |
| For further information visit                                   |
| https://www.dyalog.com/prices-and-licences.htm                 |
+-----+
Mon Oct 10 21:35:55 2022
Link Warning: [SE.Link.Create: .NET or .NetCore not available - watch defaults
to 'ns']
Linked: # -> /opt/mdyalog/Jarvis/Source
2022/10/10 @ 21:35:56 - Starting Jarvis 1.11.8
2022/10/10 @ 21:35:56 - Conga copied from /opt/mdyalog/18.2/64/unicode/ws/conga
2022/10/10 @ 21:35:56 - Local Conga reference is #.Jarvis.[LIB]
```



Docker – it can't be this easy

Run a container:

```
docker run
```

Map host port 8080 to container port 8080:

```
-p 8080:8080
```

Map host folder (where our web service code is) to /app in the container:

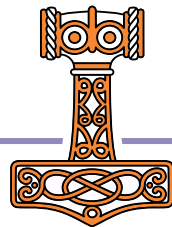
```
-v c:\sa2\v3:/app
```

Set the JarvisConfig environment variable to the location of the Jarvis configuration file:

```
-e JarvisConfig=./JarvisConfig.json5
```

Use the dyalog/jarvis public Docker container:

```
dyalog/jarvis
```

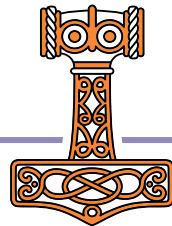


Building a Container from an Image

- Our Dockerfile

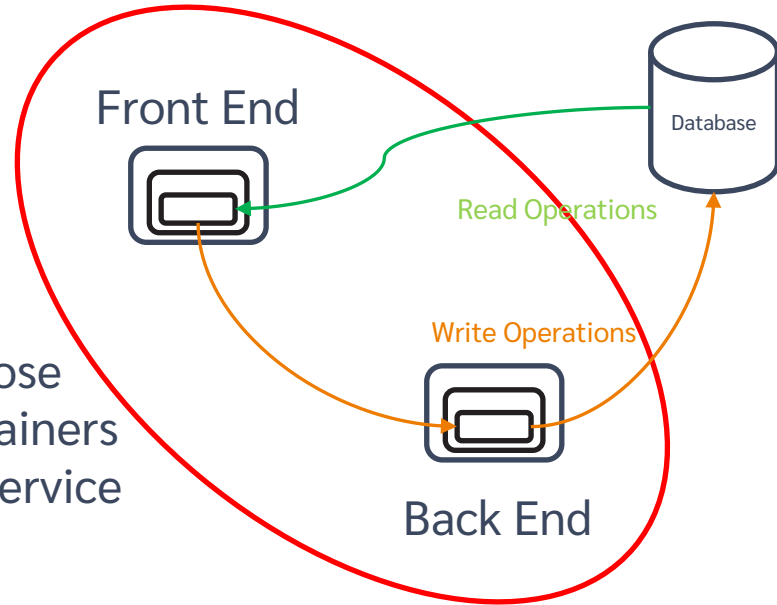
```
FROM dyalog/jarvis:latest  
ADD ./app /app  
ADD ./HttpCommand.dyalog /opt/mdyalog/Jarvis/Source
```

- Build the image and tag it as "phonebook"
`docker build -t "phonebook" .`

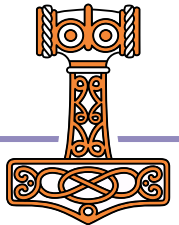


Split into Front and Back Ends

We'll call this "Two-Tier"

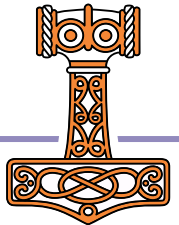


Docker Compose
defines the containers
that make up a service



Docker Compose

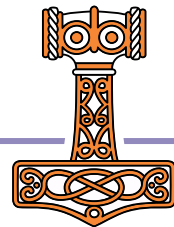
- Describes a collection of container images that make up a service.
- Creates a Virtual IP network that connects the images so that they can refer to each other by name.
- Supports replication of images and load balancing (more on this later)



Docker Compose

Two Services

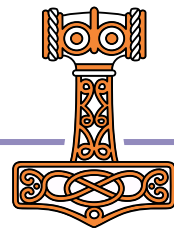
```
services:
  frontend:
    image: phonebook
    volumes:
      - ./phonebook-data:/phonebook
    ports:
      - 8080:8080
      - 8088:8088
    environment:
      - JarvisConfig=/app/frontend.json
      - DYALOG_JARVIS_THREAD=DEBUG
      - RIDE_INIT=HTTP:*:8088
  backend:
    image: phonebook
    restart: always
    volumes:
      - ./phonebook-data:/phonebook
    ports:
      - 8089:8089
    environment:
      - JarvisConfig=/app/backend.json
      - DYALOG_JARVIS_THREAD=DEBUG
      - DYALOG_JARVIS_PORT=8081
      - RIDE_INIT=HTTP:*:8089
```



Docker Compose

Two Services
Same Image

```
services:
  frontend:
    image: phonebook
    volumes:
      - ./phonebook-data:/phonebook
    ports:
      - 8080:8080
      - 8088:8088
    environment:
      - JarvisConfig=/app/frontend.json
      - DYALOG_JARVIS_THREAD=DEBUG
      - RIDE_INIT=HTTP:*:8088
  backend:
    image: phonebook
    restart: always
    volumes:
      - ./phonebook-data:/phonebook
    ports:
      - 8089:8089
    environment:
      - JarvisConfig=/app/backend.json
      - DYALOG_JARVIS_THREAD=DEBUG
      - DYALOG_JARVIS_PORT=8081
      - RIDE_INIT=HTTP:*:8089
```



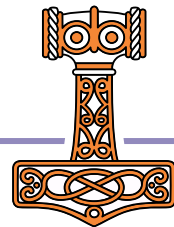
Docker Compose

```
services:
  frontend:
    image: phonebook
    volumes:
      - ./phonebook-data:/phonebook
    ports:
      - 8080:8080
      - 8088:8088
    environment:
      - JarvisConfig=/app/frontend.json
      - DYALOG_JARVIS_THREAD=DEBUG
      - RIDE_INIT=HTTP:*:8088
  backend:
    image: phonebook
    restart: always
    volumes:
      - ./phonebook-data:/phonebook
    ports:
      - 8089:8089
    environment:
      - JarvisConfig=/app/backend.json
      - DYALOG_JARVIS_THREAD=DEBUG
      - DYALOG_JARVIS_PORT=8081
      - RIDE_INIT=HTTP:*:8089
```

Two Services

Same Image

Same Permanent Storage



Docker Compose

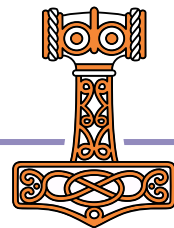
```
services:
  frontend:
    image: phonebook
    volumes:
      - ./phonebook-data:/phonebook
    ports:
      - 8080:8080
      - 8088:8088
    environment:
      - JarvisConfig=/app/frontend.json
      - DYALOG_JARVIS_THREAD=DEBUG
      - RIDE_INIT=HTTP:*:8088
  backend:
    image: phonebook
    restart: always
    volumes:
      - ./phonebook-data:/phonebook
    ports:
      - 8089:8089
    environment:
      - JarvisConfig=/app/backend.json
      - DYALOG_JARVIS_THREAD=DEBUG
      - DYALOG_JARVIS_PORT=8081
      - RIDE_INIT=HTTP:*:8089
```

Two Services

Same Image

Same Permanent Storage

Different Jarvis Configurations



Docker Compose

```
services:
  frontend:
    image: phonebook
    volumes:
      - ./phonebook-data:/phonebook
    ports:
      - 8080:8080
      - 8088:8088
    environment:
      - JarvisConfig=/app/frontend.json
      - DYALOG_JARVIS_THREAD=DEBUG
      - RIDE_INIT=HTTP:*:8088
  backend:
    image: phonebook
    restart: always
    volumes:
      - ./phonebook-data:/phonebook
    ports:
      - 8089:8089
    environment:
      - JarvisConfig=/app/backend.json
      - DYALOG_JARVIS_THREAD=DEBUG
      - DYALOG_JARVIS_PORT=8081
      - RIDE_INIT=HTTP:*:8089
```

Two Services

Same Image

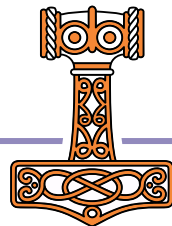
Same Permanent Storage

Different Jarvis Configurations

frontend

clients access via port 8080

Zero-footprint RIDE available on port 8088



Docker Compose

```
services:
  frontend:
    image: phonebook
    volumes:
      - ./phonebook-data:/phonebook
    ports:
      - 8080:8080
      - 8088:8088
    environment:
      - JarvisConfig=/app/frontend.json
      - DYALOG_JARVIS_THREAD=DEBUG
      - RIDE_INIT=HTTP:*:8088
  backend:
    image: phonebook
    restart: always
    volumes:
      - ./phonebook-data:/phonebook
    ports:
      - 8089:8089
    environment:
      - JarvisConfig=/app/backend.json
      - DYALOG_JARVIS_THREAD=DEBUG
      - DYALOG_JARVIS_PORT=8081
      - RIDE_INIT=HTTP:*:8089
```

Two Services

Same Image

Same Permanent Storage

Different Jarvis Configurations

frontend

clients access via port 8080

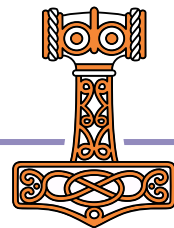
Zero-footprint RIDE available on port 8088

backend

internal only access via port 8081

Zero-footprint RIDE available on port 8089

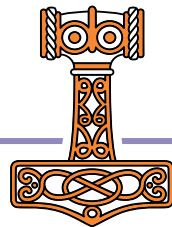
restart if the container crashes



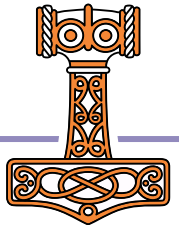
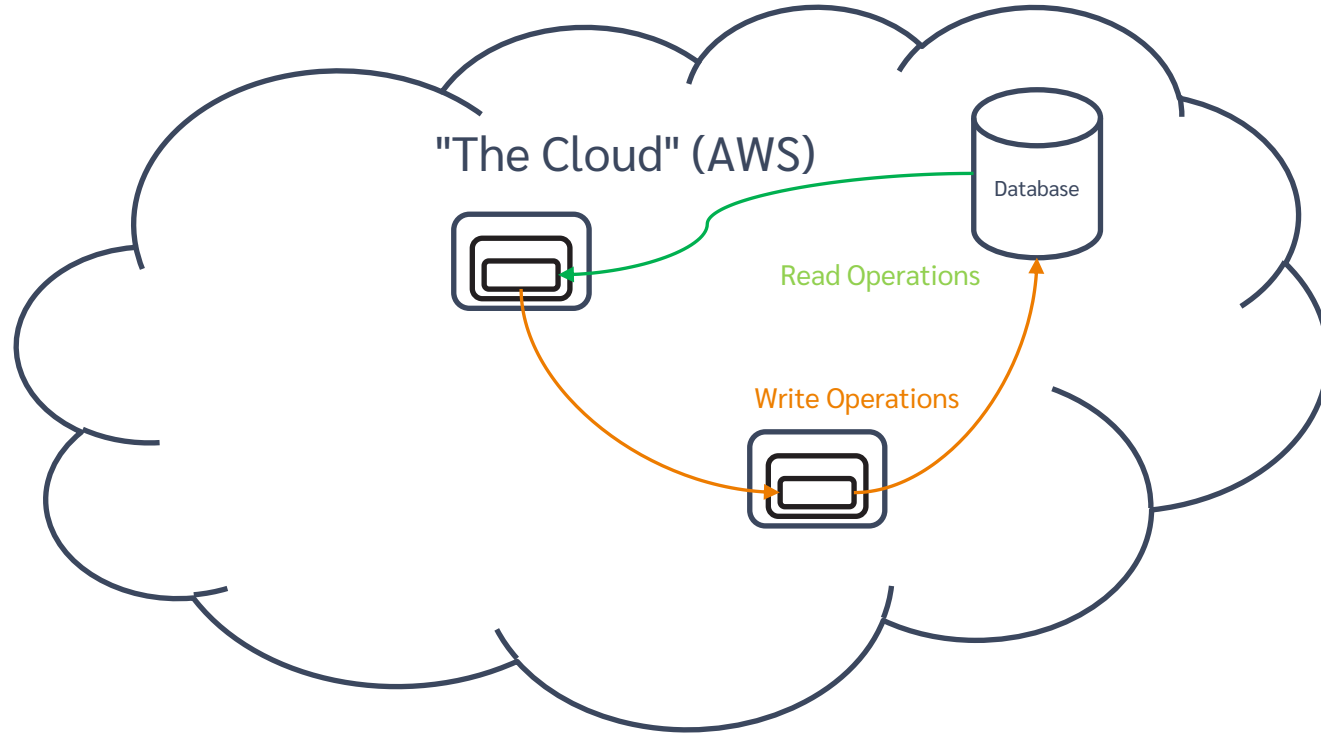
Running it locally

```
docker compose up -f docker-compose-local.yml
```

- Yes, it really is that easy...

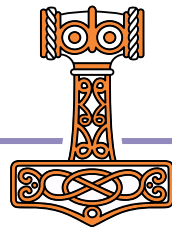


Try it in the cloud



Amazon Web Services (AWS)

- Acquire an AWS account (start with the free tier)
 - Provide credit card information
 - Cost to "experiment" is minimal



Amazon Web Services (AWS)

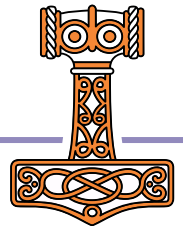
The screenshot shows the AWS Billing Management Console interface. The main content area is titled "AWS Billing Dashboard" and includes the following sections:

- AWS summary:** A grid of six summary cards.
 - Current month's total forecast: **USD 0.14**
 - Current MTD balance: **USD 0.09**
 - Prior month for the same period with trend: No data to display, **↓ 0.0%**
 - Total number of active services: **7**
 - Total number of active AWS accounts: **1**
 - Total number of active AWS Regions: **2**
- Highest cost:** A table showing the highest service spend, currently set to "Highest service spend".

Service name	Trend compared to prior month	Current MTD balance	Prior month for the same period
Elastic Load Balancing	No data to display	USD 0.08	No data to display

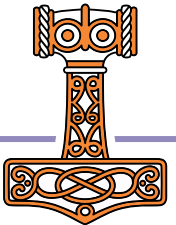
[View your bill](#)
- Cost trend by top five services:** A section for viewing data over a period of 3 months, currently set to "Last 3 months by service".

The left sidebar contains navigation links for Billing, Bills, Payments, Credits, Purchase orders, Cost & Usage Reports, Cost Categories, Cost allocation tags, Free Tier, Billing Conductor, Cost Management, Cost Explorer, Budgets, Budgets Reports, Savings Plans, Preferences, Billing preferences, Payment methods, Consolidated billing, and Tax settings.



AWS Services

- If you look the list of services provided by AWS, you'll find (at least I did) a dizzying number of choices, many with similar names...



AWS Management Console

Search for services, features, blogs, docs, and more [Alt+S]

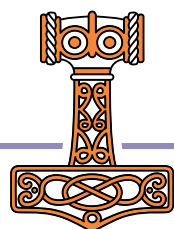
All services

Services by category

- Compute**
 - EC2
 - Lightsail
 - Lambda
 - Batch
 - Elastic Beanstalk
 - Serverless Application Repository
 - AWS Outposts
 - EC2 Image Builder
 - AWS App Runner
- Containers**
 - Elastic Container Registry
 - Elastic Container Service
 - Elastic Kubernetes Service
 - Red Hat OpenShift Service on AWS
- Storage**
 - S3
 - EFS
 - FSx
 - S3 Glacier
 - Storage Gateway
 - AWS Backup
 - AWS Elastic Disaster Recovery
- Quantum Technologies**
 - Amazon Braket
- Management & Governance**
 - AWS Organizations
 - CloudWatch
 - AWS Auto Scaling
 - CloudFormation
 - Config
 - OpsWorks
 - Service Catalog
 - Systems Manager
 - AWS AppConfig
 - Trusted Advisor
 - Control Tower
 - AWS License Manager
 - AWS Well-Architected Tool
 - AWS Health Dashboard
 - AWS Chatbot
 - Launch Wizard
 - AWS Compute Optimizer
 - Resource Groups & Tag Editor
 - Amazon Grafana
 - Amazon Prometheus
 - AWS Proton
 - AWS Resilience Hub
- Security, Identity, & Compliance**
 - IAM
 - Resource Access Manager
 - Cognito
 - Secrets Manager
 - GuardDuty
 - Inspector
 - Amazon Macie
 - IAM Identity Center (successor to AWS Single Sign-On)
 - Certificate Manager
 - Key Management Service
 - CloudHSM
 - Directory Service
 - WAF & Shield
 - AWS Firewall Manager
 - Artifact
 - Security Hub
 - Detective
 - AWS Signer
 - AWS Network Firewall
 - AWS Audit Manager
- AWS Cost Management**
 - AWS Cost Explorer
 - AWS Budgets
 - AWS Marketplace Subscriptions

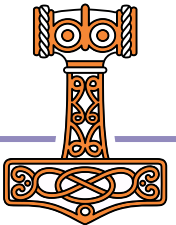
© 2022, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

You'll find
with



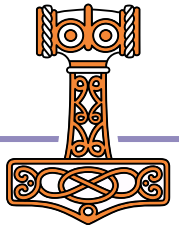
AWS Services

- If you look the list of services provided by AWS, you'll find (at least I did) a dizzying number of choices, many with similar names...
- Fortunately, Morten, with help from Google, Norbert and Bjørn worked out the right combination for our service.



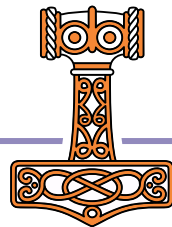
AWS Configuration

- Create an IAM user (Identity and Access Management)
 - Select Access Key credential type
 - Attach the AdministratorAccess policy
 - Download your credentials (in a .csv file)
- Install the AWS CLI (Command Line Interface)
 - Google "install aws cli"
 - Download for your platform
 - Configure to use the IAM credentials from the above step



Docker/ECS

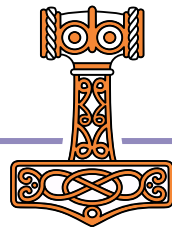
- ◆ Docker Compose can be connected to the Amazon Elastic Container Service (ECS)
 - ◆ Create a "docker context" for ECS
docker context create ecs phonebook
 - ◆ Switch to the ECS context
docker context use phonebook



Upload Image to ECR

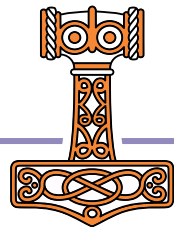
- ◆ To use our phonebook container from AWS, we need to store it either in DockerHub or the Amazon Elastic Container Registry (ECR).
- ◆ Since we already had a userid on AWS, we used ECR.
`aws ecr create-repository --repository-name phonebook`
- ◆ Now upload the image
 - ◆ Logon to the ECR server using your AWS credentials
 - ◆ `docker tag` the local phonebook image on the ECR server as `[youruserid]/phonebook`
 - ◆ `docker push [youruserid]/phonebook`

Or just edit and run the push.bat file in the workshop materials.

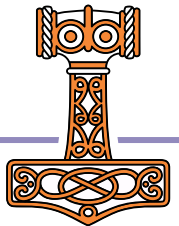
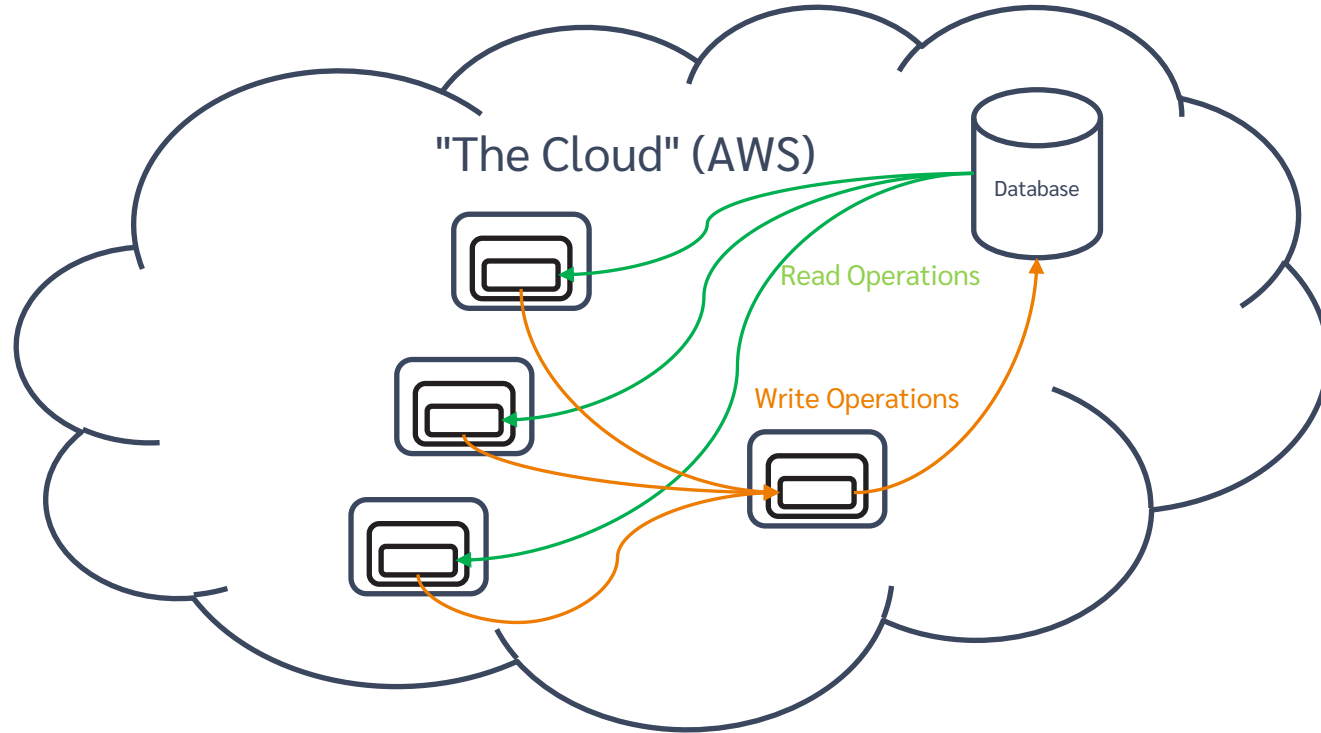


docker compose for AWS

```
services:
  frontend:
    image: [youruserid]/phonebook
    volumes:
      - phonebook-data:/phonebook
    ports:
      - target: 8080
        published: 8080
        x-aws-protocol: http
    environment:
      - JarvisConfig=/app/frontend.json
  backend:
    image: [youruserid]/phonebook
    volumes:
      - phonebook-data:/phonebook
    restart: always
    environment:
      - JarvisConfig=/app/backend.json
      - DYALOG_JARVIS_PORT=8081
volumes:
# This will be created as an "Elastic File System"
  phonebook-data:
    driver_opts:
      uid: 0
      gid: 0
```



Scale it up

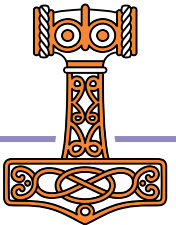


Scaling the Front End

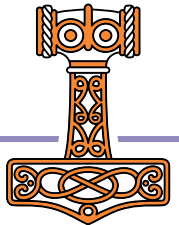
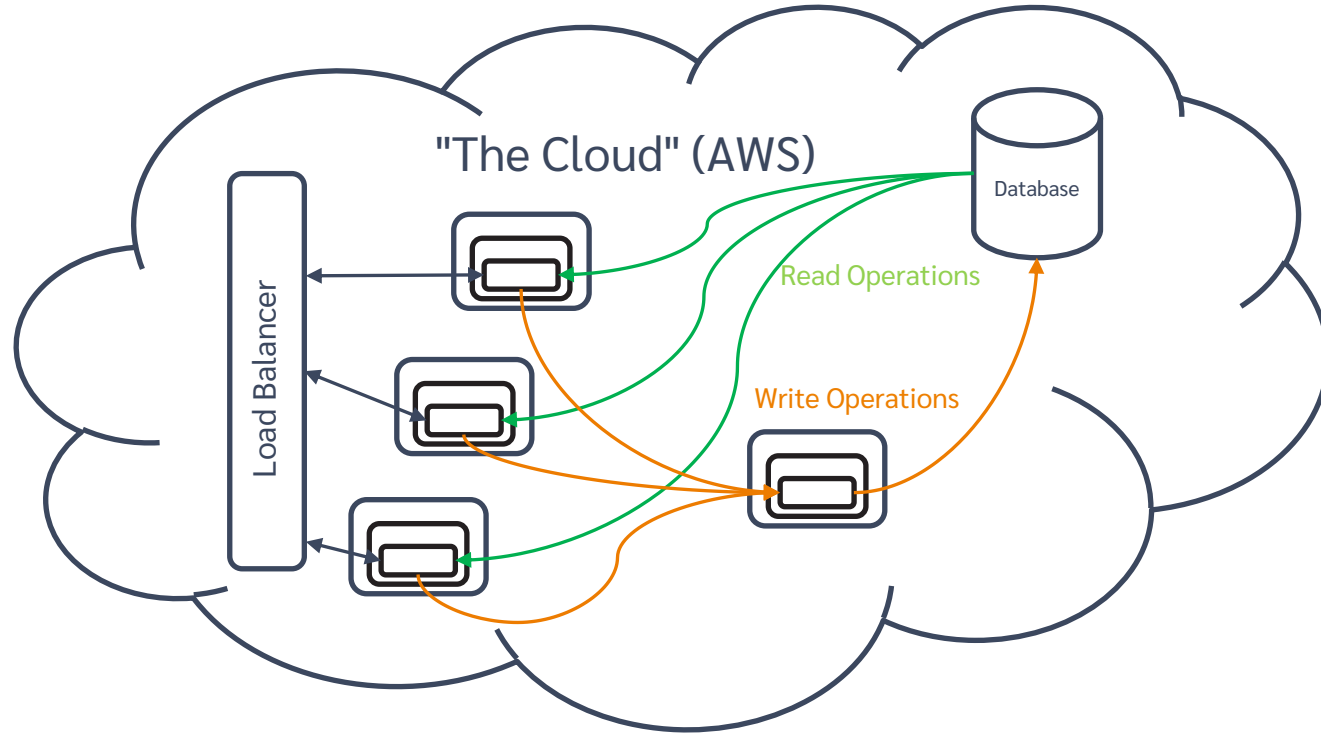
- Want 2 copies of the frontend?

```
docker compose -p phonebook up --scale frontend=2
```

- You can change the scaling while the service is running

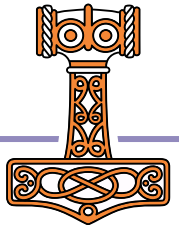


Load balance it

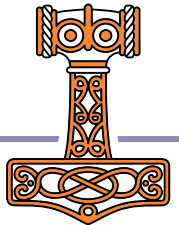
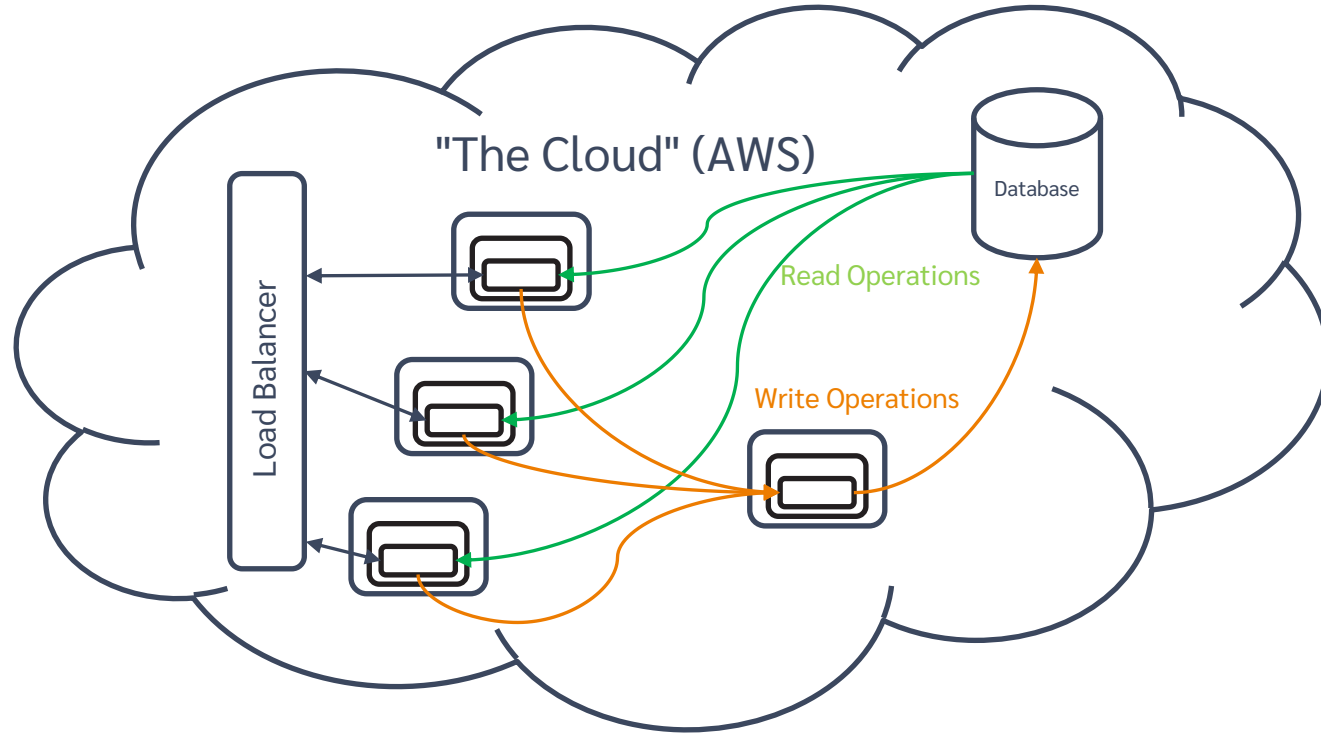


Making a Load Balancer

- ◆ This is a bit complicated...
 - ◆ Create an AWS security group
 - ◆ Add an "Ingress" for the ports you want clients to attach to
 - ◆ Finally create the load balancer itself
 - ◆ Fortunately, Morten has written a `MakeLoadBalancer` function that does all of this.
 - ◆ All you need to do is add an `x-aws-loadbalancer` entry to the docker compose file

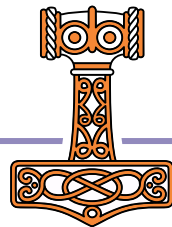


Own it

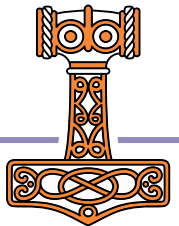
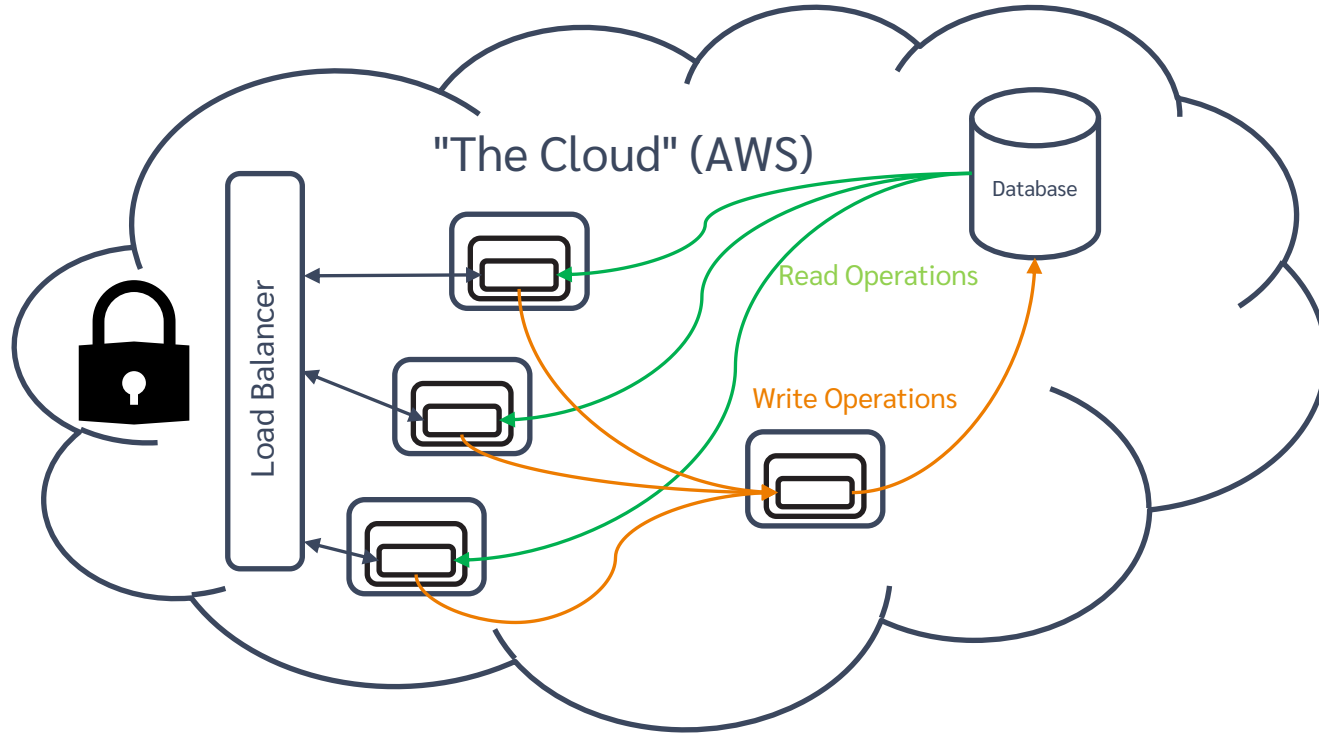


Using your own Domain Name

- ◆ Up until now, we've had to refer to the service using a long and cumbersome hostname like
`http://phone-loadb-1guewkd0evw2h-887267469.eu-west-3.elb.amazonaws.com`
- ◆ If you register your own domain with an ISP allows you to do redirection, you can have a name more to your liking.
- ◆ Morten used one.com, Brian used GoDaddy.com

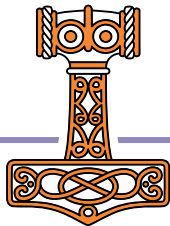


Secure it



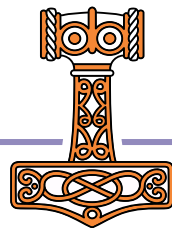
Securing your service using HTTPS

- Get a certificate – we used AWS Certificate Manager
 - ACM uses a couple simple steps to verify that you own your domain
- In AWS EC2 > Load Balancers
 - Add a listener on port 443



In Summary

- ◆ We learned a lot in preparing these workshops. We expect to learn more.
- ◆ Have we simplified the process?
 - ◆ Jarvis makes it fairly easy to turn your APL code into a web service.
 - ◆ Docker is almost ridiculously easy. (I pinch myself every time I use it)
 - ◆ AWS seems daunting, but if you know the services/components to use, it's actually pretty straightforward. And we've shown you a path to get there.



In the near future

- ◆ Updated workshop materials on [GitHub.com/dyalog-training](https://github.com/dyalog-training)
- ◆ Jarvis
 - ◆ Documentation completion (Nov 2022)
 - ◆ Training materials, samples, templates, webcasts
- ◆ Webcast series breaking down the workshops into "bite-sized" chunks
- ◆ We used AWS for the workshop; there are other services that can run Docker containers in the cloud – Azure, Google, IBM. We may publish similar guides for other services if our clients need it.

