Publish and Share

External Deployment on Amazon EC2

Version 1.0
2019-08-14

Email: contact@its4land.com
Web: https://its4land.com/

Table of Contents

Introduction

This document describes the architecture and instructions to set up the Publish and Share platform on the Amazon Web Services Cloud. The contents are aimed at system administrators and platform operators.

It is advantageous for the reader to have knowledge of basic concepts of system administration and networking using Linux. We have standardized on Ubuntu Linux 18.04 (Bionic Beaver) release for the platform. While other flavours of Linux may be used, it is left to the reader to recognize the differences and adapt accordingly. Readers are also encouraged to read up on the following to understand and customize the platform deployment as per their needs:

- **Nginx** (https://www.nginx.com/) - A highly customizable and scalable web server which has features such as reverse proxying
- **Systemd** (https://freedesktop.org/wiki/Software/systemd/) - A service manager for Linux systems to define, start, stop and log services
- **Docker** (https://www.docker.com/): Deploy and manage software containers
- **Amazon Web Services** (https://aws.amazon.com/): A suite of cloud services from Amazon to deploy and manage web applications. We use the following services:
    - **Elastic Compute Cloud (EC2)**: Set up Virtual Machine instance and scale them as per demand
    - **Elastic Block Storage (EFS)**: Block file storage service for use with EC2
    - **Elastic File Storage (EFS)**: Persistent scalable file storage for shared access between EC2 resources
    - **Simple Storage Service (S3)**: Store files as objects
    - **Relational Database Service (RDS):** Setup and manage popular DBMS on the cloud. We use a PostgreSQL instance
    - **VPC (Virtual Private Cloud)**: Setup a virtual network for AWS resources in use and define network access policies
    - **IAM (Identity and Access Manager)**: Manager user and group permission access to AWS resources
    - **Cognito**: Add and manage application user identities

Reading the Guide

This is meant to be a practical hands-on guide to work with the platform. The following visual aids are used in the document:

```
This is a code block. It denotes commands to try out, the output observed upon
running a command or textual content for configuration etc.
```

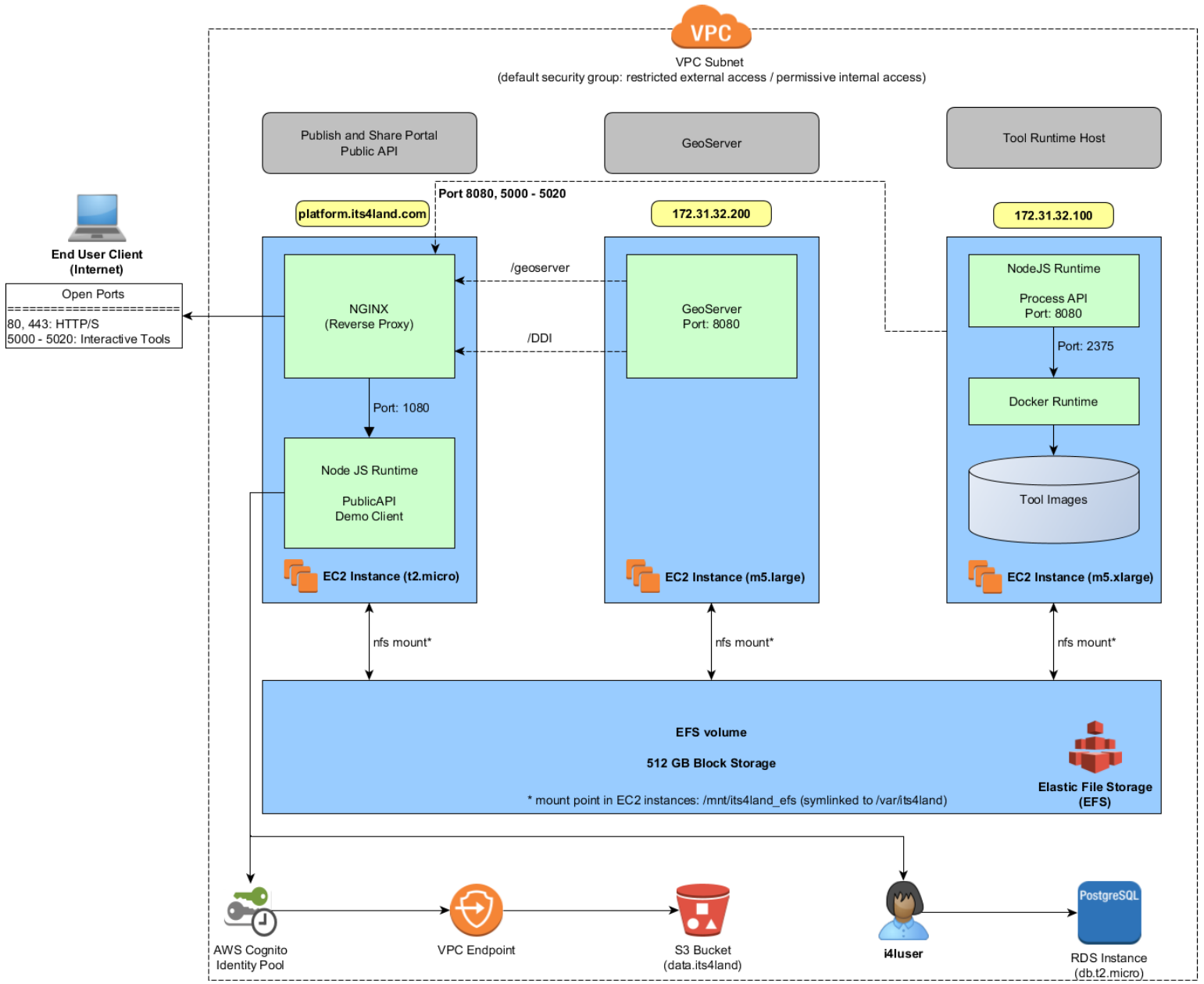
This is a note. It denotes special points to pay attention to or behaviour to expect when using a platform

This is a tip. It is intended to offer helpful pointers to the reader

This is an info block and contain additional useful information for the reader

System Architecture

The following diagram depicts the overall architecture of the system deployed on AWS.


VPC
VPC Subnet
(default security group: restricted external access / permissive internal access)
Publish and Share Portal
Public API
GeoServer
Tool Runtime Host
Port 8080, 5000 - 5020
platform.its4land.com
172.31.32.200
172.31.32.100
End User Client
(Internet)
Open Ports
80, 443: HTTP/S
5000 - 5020: Interactive Tools
NGINX
(Reverse Proxy)
/geoserver
/DDI
GeoServer
Port: 8080
NodeJS Runtime
Process API
Port: 8080
Port: 2375
Docker Runtime
Tool Images
Port: 1080
Node JS Runtime
PublicAPI
Demo Client
EC2 Instance (t2.micro)
EC2 Instance (m5.large)
EC2 Instance (m5.xlarge)
nfs mount*
nfs mount*
nfs mount*
EFS volume
512 GB Block Storage
* mount point in EC2 instances: /mnt/its4land_efs (symlinked to /var/its4land)
Elastic File Storage
(EFS)
AWS Cognito
Identity Pool
VPC Endpoint
S3 Bucket
(data.its4land)
i4luser
PostgreSQL
RDS Instance
(db.t2.micro)

The main blocks of the system consist of EC2 instances running the applications, an EFS volume for network attached storage, an S3 bucket for storing files uploaded by users and an RDS Postgresql instances to store Publish and Share platform data, and network policies which define security measures to prevent unauthorized external access to resources. An overview of each with respect to the system architecture are described next. Configuring each service from scratch will be described in the later sections.

EC2 instances

Three EC2 instances of varying sizes are used. All instances host a virtual machine running Ubuntu Linux version 18.04. The details of each of the instances are described in the following table.

Instance	Size	Network Details	Purpose and Description

Publish and Share Public API host
(main host)
	t2.micro
	8 GB primary storage
		• Uses Elastic IP for Publicly visible IPv4 access
		• Accessible on **platform.its4land.com**
		• Open ports:
		  ○ 80/443: HTTP/S access
		  ○ 5000-5020: access web apps for interactive tools
			• Main host for the Publish and Share platform
			• Hosts the portal page resources and Public API
			• Nginx is used as a reverse proxy server and forwarding of requests to corresponding hosts
			• Port and route forwarding details:
			  ○ :80 → localhost:8080
			  ○ /geoserver → 172.31.32.200:8080/geoserver
			  ○ /DDI → 172.31.32.200:8080/geoserver/its4land
			  ○ :8080 → 172.31.32.100:8080

Geoserver host	*m5.large** 8 GB primary storage	• Primary IPv4 address uses DHCP • Secondary IPv4 address bound to **172.31.32.200**	• **Hosts geoserver on port 8080** • Host accessible from within subnet only. • External access via */geoserver* route on the main host
Process API host	*m5.xlarge** 64 GB primary storage	• Primary IPv4 address uses DHCP • Secondary IPv4 address bound to **172.31.32.100**	• **Hosts tools for Publish and Share platform** • ProcessAPI runs on port 8080 and Docker runtime API on port 2375 • ProcessAPI accessible from within VPC subnet only. • Docker containers for tools run here • Only interactive tools which run applications in the range 5000-5020 are externally accessible using reverse proxy on the main host

*large instances are more powerful in terms of memory and computing power, but cost more to run and should be stopped when not in use.

Elastic File Storage Volume

EFS allows for storage to be network-attached and shared across several instances. Usage is billed based on the amount of storage used. While we could have added separate block storage (using EBS), for each instances, EBS volumes cannot be mounted simultaneously across instances. Besides, a single EFS volumes allows for easier backup and sharing of files if needed between the EC2 instances.

Volumes using EFS maybe mounted as a networked file system (NFS). The efs utils package (https://github.com/aws/efs-utils) offered by Amazon eases this process.

For Publish and Share, in each EC2 instance the EFS volume is mounted at the location: `/mnt/its4land_efs` . This mount point is then symlinked to `/var/its4land` for convenience.

S3 Store

S3 provides a scalable object storage web service. Data such as files can be stored along with their attributes and retrieved via AWS APIs. The role of S3 in Publish and Share is to act as the repository for content uploaded by end users. While EFS can also be used, the advantages offered by S3 make it a better choice for this purpose:

- better scalability, versioning, backup and restore
- per object logging allowing us to track which ones are accessed more frequently
- higher performance than EFS (avoids NFS bottlenecks)
- cheaper pricing per gigabyte
- more flexible security policies via access control lists and bucket policies
- ideal for storing static files if an underlying filesystem is not needed

We use the bucket **data.its4land** to store uploaded content items. Each uploaded content item is renamed and stored as an object identified by its UUID. The Publish and Share database table *t_ContentIndex* stores metadata about the original file.

The security features are configured in the following manner:

- The Public API uses temporary credentials generated by the AWS Cognito service. The credentials belong to the 'Publish and Share' *Identity Pool.*
- The bucket policy for *data.its4land* is configured for credentials belonging to the identity pool so that:
    - only traffic originating from within the VPC subnet and with the allowed credentials is allowed read/write access
    - direct read/write access to the bucket objects is not allowed from external (internet) sources

    - read/write access is allowed only via the Public API `/contentitems` endpoint
- We use a VPC endpoint to route requests to S3 originating from the VPC subnet internally, instead of going through the internet
- Bucket owners have a more permissive policy allowing them direct read/write access including managing of bucket policies

RDS

AWS RDS provides a relational database management system on the cloud, supporting several popular databases. The details of the db instance used are as follows:

- Database: PostgreSQL (version 11)
- Instance Type: db.t2.micro
- Configuration: 1 vcpu, 1 GB RAM, 20 GB (autoscaling upto 1000 GB)

The instance is accessible from within the VPC network and external access is disallowed.

We use this instance to initialize the `i4l` database to store data pertaining to the Publish and Share platform. The table below shows the users.

Role Name	Role	Description
i4ldba	Database Administrator	Has administrator permissions allowing the creation of schemas, tables and other database artifacts including creating roles and users
i4luser	User	Has read-write permissions on tables belonging to the `i4ldata` schema and to the schema itself

Network & Security

The AWS account to host the Publish and Share platform is hosted in the EU (Frankfurt) region. The account comes with a default virtual private cloud (VPC) which comes with its own IPv4 subnet, routing table. Access to the network can be controlled via access control lists and security groups. We stick with default settings for the former and use the latter to restrict access outside of the subnet to selected resources and network ports. Access within the subnet is permissive provided proper credentials are used (for S3/RDS) or services are accessed on allowed ports (e.g geoserver on port 8080).

The following table summarizes the network security policy for incoming requests:

AWS Resource	Security Policy
Main EC2 instance	• Uses Nginx reverse proxy to forward requests to upstream hosts • External access to Public API and portal page on port 80

AWS Resource	Security Policy
	• External access to geoserver on `/geoserver` route • External access to web based tools started on the ProcessAPI host on ports 5000-5020 • All other ports (VPC subnet access)
Geoserver EC2 instance	• Only VPC subnet access
ProcessAPI EC2 instance	• Only VPC subnet access
EFS volume	• Only VPC subnet access
S3 store	• Direct access to objects in the `data.its4land` bucket only for valid Identity pool credentials for requests that come in via the VPC endpoint • External access only via the `/contentitems` endpoint
RDS instance	• Only VPC subnet access for users with proper database credentials

Everytime an EC2 instance is started, it acquires a random private (accessible in the subnet) IPv4 address via DHCP. Since a host IP address is required for accessing services which run on an instance such as GeoServer, we assign a secondary static IP address to selected instances. The addresses **172.31.32.100** and **172.31.32.200** are used for the ProcessAPI and Geoserver EC2 instances respectively. This ensures the hosts are accessible on a fixed address even when the instances are stopped and started.

The main EC2 instance needs to be publicly accessible. To have a fixed IP address, we have used an *Elastic IP* and assigned it to the instance.

While the security policies described above deal only with end user client requests, developers will require a more permissive set of policies for ease of deployment and debugging. For this reason, custom security groups have been defined for the resources to allow direct access for custom IPv4 ranges. The rules allow selected machines e.g. machines in Hansa Luftbilds's network to access the resources directly. These are useful to access services such as SSH, SFTP and to test and debug the deployed web services, databases etc.

Users with proper permissions can view and edit these rules from the AWS console (https://eu-central-1.console.aws.amazon.com/console/home?region=eu-central-1)

Deployment Guide

In this section we will go through the steps required to setup and configure AWS resources to host the its4land Publish and Share platform. All resources in Publish and Share are currently hosted in the *EU (Frankfurt)* region.

Detailed instructions of how to setup an AWS resource instance or service **will not be covered here**. If instructed to 'set up an EC2 *m5.large* instance', we expect the reader to be familiar with or to refer to AWS documentation and guides to set one up. This is left as an exercise for the reader. Rather we will mention the required configuration for the resource/service to be setup and the next steps to be followed in order to setup the platform itself.

When creating AWS resource instances or services using wizards, just choose the default option if you are unsure of which option to select

The assumption in the deployment guide is that the user has an AWS account and necessary permissions to setup the resources. If not, the user needs to contact the organization's root account holder or other superusers who are able to grant the identity and permissions

Setting up an RDS instance

Create the instance

Create a database using the wizard. For its4land, we choose the following configuration:

- PostgreSQL (version 10 or higher)
- DB instance identifier: `i4ldb`
- Template: free tier
- Master username: `i4ldba`
- Set the master password. We will use `mydbapassword` for demonstration (this is a bad password and you should use best practices to choose one).
- Connectivity > Additional configuration: Set publicly accessible to yes
- Other options remain default

Once the database has been created, check and configure the security group rules so as to allow connecting to the instance endpoint shown in the console. Also add it to the VPC's default security group to permit access within the subnet.

The instance endpoint name will be of the form: `i4ldbinstance.df7u335q0adm.eu-central-1.rds.amazonaws`

Setup the database in the created instance

To connect to the newly created instance, we will use the `psql` tool. This tool is available in the `postgresql-client` package in Ubuntu. For Windows, one can obtain it from the official Postgresql binaries (https://www.postgresql.org/download/windows/).

The SQL DDL scripts to setup the database structure and add sample data for selected tables are available in the *its4land* git repository in the *SetupDB* branch. To initialize the database on the newly created RDS instance:

- Edit the file `<repository_root>/PostgreSQL/setup/ddl/init_Roles.sql` and change the line shown below to set the password for the newly created user `i4luser`. This is the user that will have limited read/write access to the database. Save the file once you change the password.

```
CREATE ROLE i4luser WITH LOGIN IN ROLE i4l_readwrite PASSWORD 'mypassword';
```

- Run the following from the root directory of the repository (`psql` must be available in the PATH).

```
cd PostgreSQL/setup
psql -h i4ldbinstance.df7u335q0adm.eu-central-1.rds.amazonaws -d postgres -U
i4ldba
```

- Enter the password i.e. `mydbapassword`, when prompted for one after running the above command
- If the login was successful, you should see be able see a psql prompt (it will be something like: **postgres=#**) . On this the prompt type:

```
\i i4lSetup.sql
```

- This will create the user `i4luser`, add a database schema `i4ldata`, add tables to this schema, and add data to a few select tables.
- To check if the tables have been created on the psql prompt, type:

```
\dt i4ldata.*
```

- This should list all the newly created tables. In the owner column, `i4luser` should be listed as the owner.
- Type `\q` to exit the prompt. To make sure that the user `i4luser` has been created, login as this user to the newly added database and enter the password added above i.e. `mypassword`.

```
psql -h i4ldbinstance.df7u335q0adm.eu-central-1.rds.amazonaws -d i4l -U i4luser
```

While we use `psql` in the examples above, the same can be accomplished by using other popular database administration utilities which support PostgreSQL such as pgAdmin or DBeaver, albeit with a few changes.

The main `i4lSetup.sql` is `psql` specific and automates the database setup by including table creation scripts in the `ddl` directory. If using a different client, the roles, table and schema will need to be created manually by running each of the SQL scripts within the `ddl` directory.

Setting up S3

Configuring the S3 service to host the Publish and Share platform involves three major steps.

1. Create separate credentials for S3 access

- Create a new identity pool using the AWS Cognito service. Make sure that '*Enable access to unauthenticated identities*' is selected. We will name the identity pool `Publish and Share`
- In the `Publish and Share` identity pool, create an unauthenticated role `- Cognito_TestIDpoolUnauth_Role`

2. Create a VPC endpoint for the default VPC

- A VPC endpoint allows supported AWS services to communicate with other resources in the VPC without the traffic leaving the Amazon network (otherwise the traffic between EC2 instance and S3 can pass through the internet)
- The creation wizard for a VPC endpoint allows choosing the service for which the endpoint is required (S3). The endpoint for S3 is a gateway endpoint.
- Let's assume that the ID of the gateway endpoint is `vpce-04193c7ebaa50de11`

3. Add an S3 bucket and set its policies

- Using the S3 management console, create a new bucket. The bucket used officially in its4land is named **data.its4land**
- Uncheck 'Block *all* public access' during bucket creation. or later in the *Permissions* tab of the bucket.
- In the *Permissions* tab, set the Bucket Policy with the following contents. (you might need replace *AdminUser*, numerical IDs in the ARN and vpc endpoint IDs with proper values as per your configuration):

```
{
    "Version": "2012-10-17",
    "Id": "its4land-s3-data-access-policy",
    "Statement": [
        {
            "Sid": "Allow-everything-for-user-AdminUser",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::680548971110:user/AdminUser"
            },
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::data.its4land",
                "arn:aws:s3:::data.its4land/*"
            ]
        },
        {
            "Sid": "Allow-read-write-for-cognito-temp-user-from-vpc",
            "Effect": "Allow",
            "Principal": {
                "AWS":
"arn:aws:iam::680498787150:role/Cognito_PublishandShareUnauth_Role"
            },
            "Action": [
                "s3:GetObject",
                "s3:GetObjectTagging",
                "s3:PutObject",
                "s3:PutObjectTagging",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::data.its4land",
                "arn:aws:s3:::data.its4land/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:sourceVpce": "vpce-04193c7ebaa50de11"
                }
            }
        },
        {
            "Sid": "VPC-endpoint-readonly-access-for-unapproved-users",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:getObject",
            "Resource": [
                "arn:aws:s3:::data.its4land",
                "arn:aws:s3:::data.its4land/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:sourceVpce": "vpce-04193c7ebaa50de11"
                }
            }
        }
    ]
}
```

- The above policy:
    a. Allows the *AdminUser* complete access to the *data.its4land* bucket regardless of the origin of request

   b. For the role created using Cognito, read-write access to the bucket is limited to traffic only originating from the VPC endpoint with ID `vpce-04193c7ebaa50de11`
   c. For other users, allow only read-only access to bucket objects for requests originating from the VPC endpoint with ID `vpce-04193c7ebaa50de11` (useful for debugging).

- The intent of the above policy is to ensure that bucket objects can only be read from or written to via the `/contentitems` endpoint of the PublicAPI.

Setting up an EFS filesystem

- Create an EFS filesystem using the creation wizard. This volume will then me mounted onto EC2 instances created later as network attached storage.
- In the creation wizard and select the default VPC and zones when creating the volume.
- After the volume has been created, you should see the newly created filesystem along with a filesystem ID. This ID will be used to mount the volume.

Setting up EC2 instances

The first step is to set up EC2 instance on which the platform is set up. To do this create an EC2 instance using creation wizard. Use the *Ubuntu 18.04 LTS - Bionic* image available in the AWS marketplace. Make sure that the created instance will be assigned a Public IPv4 address when started. Create three such instances:

- One *t2.micro* instance (with 8 GB primary storage) - we will call this the *main instance*
- One *m5.large* instance (with 8 GB primary storage) - this will be the *geoserver instance*
- One *m5.xlarge* instance (with 64 GB primary storage) - this will be the *ProcessAPI instance*

Once instances have been created, add them to the VPC's *default security group*. This will ensure one instance will be able to access services provided by the other over the VPC network.

Logging in into created instances

To log into an instance:

- The inbound security group rules for the instance should allow SSH access from 0.0.0.0/0 or the desired IP address
- The Public IPv4 address or hostname can be viewed in the AWS console
- Default username for Ubuntu instances is `ubuntu`
- For login authorization use the key pair selected during instance creation
- After making sure of the above, you can login using any standard SSH client such as Putty

In order have a fixed, publicly accessible IPv4 address for an instance, AWS EC2 provides Elastic IPs. You can obtain and Elastic IP and assign it to an instance. This way the machine will have a fixed IPv4 address, even if it is stopped and restarted later. For the officially deployed Publish and Share platform, we use an Elastic IP address for the main instance.

Configure the main instance

`tmux` is a very useful terminal multiplexer allowing multiple terminal clients to be opened in one window. It is highly recommended to install and learn how to use this package especially when working with remote logins.

Some of the steps below will require root user permissions to edit files and run commands. This can be done with `sudo` privileges for the default `ubuntu` user.

1. Install packages

Once you have logged in, now it is time to install required packages from the official Ubuntu package repository. This is done as shown below.

```
sudo apt install nginx mongodb nodejs curl
```

2. Mount the EFS filesystem

In the next step, we mount the EFS filesystem we created previously. Amazon provides the `efs-utils` package to ease mounting of file system. To install this package on Ubuntu, follow the instructions on https://github.com/aws/efs-utils. Create a Debian package following the instructions and install it.

- Check the AWS console for the EFS filesystem ID. It will be something like `fs-f40a36bc` .
- From the console, chose 'Actions > Manage file system access'. In the 'Manage mount options' screen that is shown next, If the *default security group* is not shown in the list of security groups, add it here.
- This filesystem will be mounted on a directory which we refer to as the mount point. Follow the instructions below.

```
sudo mkdir -p /mnt/its4land_efs
sudo chown ubuntu:ubuntu /mnt/its4land_efs
sudo mount -t efs fs-f40a36bc:/ /mnt/its4land_efs
```

If successfully mounted, you should see the `/mnt/its4land_efs` mount point in the list of mounted filesystems when you type `mount` on the console.

To ensure that the EFS filesystem is mounted every time the EC2 instance is booted up, add the following line to the `/etc/fstab` file using an editor such as Vim (`sudo vim /etc/fstab`)

```
fs-f40a36bc:/    /mnt/its4land_efs       efs     defaults,_netdev        0 0
```

Finally, we add a link to the mount point for convenience. This way, even if we switch to a different filesystem later, all we need to do is to redirect the link to the new mount point.

```
sudo ln -s /mnt/its4land_efs /var/its4land
```

It is possible to mount the EFS filesystem without installing the efs-utils. To do so, follow the instructions given in 'Amazon EC2 mount instructions (from local VPC)' the EFS console when you select the filesystem. This uses the standard nfs client from the official Ubuntu repositories. This method however is inconvenient and therefore not used here.

3. Add Publish and Share platform files

- Obtain a release archive containing Publish and Share PublicAPI integrated with Expermaps.

  Note for Internal developers at HL

  Instructions to build Expermaps project are here - Ein ExperMaps Projekt bauen

- Create a directory `/var/its4land/expermaps` and unzip the release archive there
- In the `expermaps` directory, edit the file `bin/startupConfigs/default.ini` . Add the following database configuration or change it if present to match the RDS configuration created previously.

  ```
  sdcPostgresDB = postgresql://i4luser:mypassword@i4ldbinstance.df7u335q0adm.eu-central-1.rds.amazonaws:5432/
  ```

- Edit the file `bin/em.js` and change the following values for HTTP and HTTPS ports. This is necessary, since we will be reverse proxying the default HTTP/S ports using Nginx.

```
httpPort = 1080;
...
httpsPort = 1443;
```

4. Setup MongoDB

The Expermaps demo client uses MongoDB. We need to set it up first.

- Stop the mongodb service if it is running.

```
# Check if service is running
systemctl status mongodb.service


# Stop a running service
sudo systemctl stop mongodb.service
```

- Create the directory `/var/its4land/mongodb_data`. Also change its ownership.

```
mkdir -p /var/its4land/mongodb_data
sudo chown -R mongodb:mongodb /var/its4land/mongodb_data
```

- Edit the file `/etc/mongodb.conf` (you will need `sudo` permissions) and change or add the following line to point to the newly created directory

```
dbpath=/var/its4land/mongodb_data
```

- Now start and enable (starts service on machine reboot) the mongodb service

```
sudo systemctl start mongodb.service
sudo systemctl enable mongodb.service
```

- Now setup the Mongodb database for Expermaps

```
cd /var/its4land/expermaps
node bin/createDB.js its4land
```

5. Test if Expermaps and Public API are running

- At this point expermaps and the Publish and Share Public API should be able to run. To test it try the following in the `/var/its4land/expermaps` directory.

```
node bin/em.js
```

- Expermaps and the Public API should start correctly. Error logs for Expermaps will be available in the `logs` directory. Errors for the PublicAPI should be shown on the standard output. To test if it they are loading correctly, try this from another console

```
# Test Expermaps
curl localhost:1080/demo


# Test PublicAPI
curl localhost:1080/api/
```

- Upon sending the requests using cURL, we expect some HTML to be shown on the standard output.
- If the main instance has a public IPv4 address and access to port 1080 is allowed in the security group, you can also use a browser to check if it working. Just replace 'localhost' by the IP address.
- Contact the developers if you face an error along with the error logs if available.

6. Setup Nginx

- The next step is to configure Nginx as the reverse proxy for the NodeJS application running on port 1080.
- First edit `/etc/nginx/nginx.conf` and add the following lines **within the** `http` **block** to add zones and limit number of connections for each zone:

```
## Security settings
limit_req_zone $binary_remote_addr zone=root_zone:50m rate=50r/s;
limit_req_zone $binary_remote_addr zone=api_zone:50m rate=100r/s;
```

- Next create the file `/etc/nginx/sites-available/pus-main.conf` and add the following contents:

```
#####################################################
### Settings for the Publish and Share platform ###
#####################################################

upstream pus_api_host {
         server 127.0.0.1:1080;
}

# Main Website + Public API
server {
    listen 80 default_server;

    limit_req zone=root_zone burst=200;

    # Reroute requests to NodeJS server instance
    location / {
        rewrite ^(/?)$ $1portal last;
        proxy_pass http://pus_api_host/;
    }

    location /demo {
        proxy_pass http://pus_api_host/;
    }

    location /api {
        limit_req zone=api_zone burst=500; # Rate limiting
        proxy_pass http://pus_api_host/api;
    }

    # Forward requests send to /sub/* to /api/*
    location /sub/ {
        rewrite ^/sub(/.*)$ /api$1 redirect;
        #rewrite ^/sub$ /api break;
        proxy_pass http://pus_api_host/api/;
    }

}
```

- Add a link to the `/etc/nginx/sites-enabled` directory. If there is a default site already configured, remove or unlink it.

```
cd /etc/nginx/sites-enabled
ln -s ../sites-available/pus-main.conf
```

- Test if the configuration is valid by running. If there are no mistakes, you should receive a message saying the test is successful. Otherwise you will need to fix the configuration error.

```
sudo nginx -t
```

- Finally, we start and enable the nginx service

```
sudo systemctl start nginx.service
sudo systemctl enable nginx.service
```

7. Create a Systemd service for Publish and Share

- If the testing is successful, our next step is to create a systemd service, so that the Expermaps and the PublicAPI are started automatically after booting.
- To create the service, create the file `/etc/systemd/system/publishandshare.service` (you will need root permissions), and add the following contents:

```
[Unit]
Description=Publish and Share platform Public API service
After=nginx.service mongodb.service

[Service]
Type=simple
User=ubuntu
WorkingDirectory=/var/its4land/expermaps
ExecStart=/usr/bin/node bin/em.js
TimeoutStartSec=90s
TimeoutStopSec=30s
RestartSec=90s
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

- Start and enable the publish and share service

```
sudo systemctl daemon-reload
sudo systemctl start publishandshare.service
sudo systemctl enable publishandshare.service
```

You can check the status of the service this way, to see if it is active and running:

`systemctl status publishandshare.service`

To monitor log entries for this service, you can use the following command:

`journalctl -xe -u publishandshare.service`

- If the security group of the main instance has been configured such that the HTTP/S ports are public, then visiting the IPv4 address in a web browser should show the default Publish and Share portal page.

Configure the Geoserver instance

1. Install Packages

- Once you have logged in, now it is time to install required packages from the official Ubuntu package repository.

```
sudo apt install openjdk-8-jre
```

2. Mount EFS filesystem

- Follow the same instructions as for the main instance.

3. Download and Configure Geoserver and Extensions

- Downloading a stable release is recommended. At the time of writing this is http://sourceforge.net/projects/geoserver/files/GeoServer/2.15.2/geoserver-2.15.2-bin.zip
- Download the S3 Geotiff plugin for the corresponding release of Geoserver. This is available from the community builds. For Geoserver 2.15.x, this can be downloaded from - https://build.geoserver.org/geoserver/2.15.x/community-latest/geoserver-2.15-SNAPSHOT-s3-geotiff-plugin.zip
- Unzip the contents of the geoserver release to `/var/its4land/geoserver-2.15.2` (or the corresponding version number).
- The S3 geotiffs plugin archive contains a number of *jar* files. Unzip this file to the `/var/its4land/geoserver-2.15.2/webapps/geoserver/WEB-INF/lib` directory.
- Create a soft link to the geoserver directory in `/usr/share`

```
sudo ln -s /var/its4land/geoserver-2.15.2 /usr/share/geoserver
```

- At this point, you can test if Geoserver is setup properly by running the following on the command line. This should start geoserver on port 8080.

```
GEOSERVER_HOME=/usr/share/geoserver /usr/share/geoserver/bin/startup.sh
```

> You should change the Geoserver default administrator password. Follow the official Geoserver documentation for instructions on how to do this

4. Create a Systemd service for Geoserver

- If successful was able to start successfully upon testing, our next step is to create a systemd service.

- To create the service, create the file `/etc/systemd/system/publishandshare.service` (you will need root permissions), and add the following contents:

```
[Unit]
Description=Geoserver for Publish and Share
After=network.target

[Service]
Type=simple
User=ubuntu
Environment=GEOSERVER_HOME='/usr/share/geoserver'
ExecStart=/usr/share/geoserver/bin/startup.sh
ExecStop=/usr/share/geoserver/bin/shutdown.sh
TimeoutStartSec=60s
TimeoutStopSec=90s
RestartSec=90s
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

- Start and enable the publish and share service

```
sudo systemctl daemon-reload
sudo systemctl start geoserver.service
sudo systemctl enable geoserver.service
```

5. Setup a Static IPv4 address

- Everytime the Geoserver EC2 instance is stopped and started, it obtains a Private IPv4 address (on the VPC subnet) via DHCP. To ensure a uniform address we need to add a static IPv4 address.
- First, **in the AWS console**, assign a secondary private IP to the geoserver instance with the value **172.31.32.200**.
- Next, we configure the IP address in the running instance. Ubuntu 18.04 uses the Netplan utility for network configuration. There should be an existing file `/etc/netplan/50-cloud-init.yaml`. Edit this file to add the address `addresses:` and `routes:` fields as shown below. The example below uses the static IP address **172.31.32.200**.

```
# This file is generated from information provided by
# the datasource.  Changes to it will not persist across an instance.
# To disable cloud-init's network configuration capabilities, write a file
# /etc/cloud/cloud.cfg.d/99-disable-network-config.cfg with the following:
# network: {config: disabled}
network:
    version: 2
    ethernets:
        ens5:
            dhcp4: true
            match:
                macaddress: 06:77:1c:f4:7c:1a
            set-name: ens5
            addresses:
              - 172.31.32.200/20
            routes:
              - to: 0.0.0.0/0
                via: 172.31.32.1 # Default gateway
                table: 1000
              - to: 172.31.32.200/20
                via: 0.0.0.0
                scope: link
                table: 1000
```

The subnet that your EC2 instance is on might be different. Your should check which subnet your EC2 machine belongs to, from the AWS console, and add a static IP in the valid CIDR range of that particular subnet.

- After editing the file, it is time to apply the changes. Use to following command to test and apply the settings if they are valid.

```
sudo netplan try
```

- If making the changes was successful, you can test whether the Geoserver instance is reachable from the main instance by pinging 172.31.32.200 from the main instance. It should be ping-able in order for reverse proxying on nginx, described next.

Setting the hostname (optional)

- To address the machine by a name instead of its IP address, do the following

```
sudo hostnamectl set-hostname geoserver.its4land.com
```

- Add/edit entry in `/etc/hosts`

```
127.0.0.1 geoserver.its4land.com geoserver localhost
```

6. Configure Nginx on main instance to add geoserver route

- Our final step in completing the configuration is to add a route for Geoserver on Nginx. Edit the `/etc/nginx/sites-available/pus-main.conf` **on the main instance,** and add the following *within the* `server` *block*:

```
location /DDI {
    proxy_pass http://172.31.32.200:8080/geoserver/its4land;
}

location /geoserver {
    proxy_pass http://172.31.32.200:8080/geoserver;
}
```

- Test and Nginx configuration and restart the service as described previously
- If successful, geoserver should now be reachable on `http://<public ip address of main instance>/geoserver`

Configure the ProcessAPI instance

1. Install Packages

- Login and install required packages from the official Ubuntu package repository.

```
sudo apt install nodejs redis docker npm
```

2. Mount EFS filesystem

- Follow the same instructions as for the main instance.

3. Configure Docker

- Add user `ubuntu` to the `docker` group (this will allow the user to run docker containers without using sudo)

```
sudo gpasswd -a ubuntu docker
```

- Allow access to the Docker daemon on port 2375. To do this, override the default Docker service's command line parameters by creating the file `/etc/systemd/system/docker.service.d/override.conf` with the following contents.

```
[Service]
ExecStart=
ExecStart=/usr/bin/dockerd -H fd:// -H tcp://0.0.0.0:2375 --
containerd=/run/containerd/containerd.sock
```

- Restart and enable Docker service

```
sudo systemctl daemon-reload
sudo systemctl restart docker.service
sudo systemctl enable docker.service
```

- Test if Docker engine API is accessible. If successful, you should see version information of the installed Docker build in JSON format on stdout.

```
curl http://localhost:2375/version
```

4. Start and enable Redis service

- Redis is an in-memory store and can also be used as a message broker. It is used by the ProcessAPI to track and update processes on the Publish and Share platform. Start and enable the Redis service using systemd.

```
sudo systemctl start redis.service
sudo systemctl enable redis.service
```

5. Setup Process API

- Obtain ProcessAPI from the git repository. Copy the entire `<repository_root>/ProcessAPI` directory to `/var/its4land/ProcessAPI`
- Start the ProcessAPI

```
cd /var/its4land/ProcessAPI
npm install
node index.js
```

- Test if the API is functioning as expected. You should receive a HTTP 200 OK status upon running the command below.

```
curl -I localhost:8080/processapi/v1/ping
```

6. Add a Process API service

- If the ProcessAPI test was successful, create a systemd service.
- To create the service, create the file `/etc/systemd/system/processapi.service` and add the following contents:

```
[Unit]
Description=Process API service for the Publish and Share platform
After=redis.service docker.service

[Service]
Type=simple
User=ubuntu
WorkingDirectory=/var/its4land/ProcessAPI
ExecStart=/usr/bin/node index.js
TimeoutStartSec=90s
TimeoutStopSec=30s
RestartSec=90s
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

- Start the service

```
sudo systemctl daemon-reload
sudo systemctl start processapi.service
sudo systemctl enable processapi.service
```

7. Configure the Static IP address

- Configure a static private IP address within the subnet. Follow the same instructions as with the geoserver instance but instead using the IP address **172.31.32.100**

8. Adding Docker images

A a tool writer can provide a Docker image to the administrator in two ways:

1. As a source archive containing the Dockerfile (https://docs.docker.com/engine/reference/builder/) along with any necessary scripts & executable files. This requires that any parent image or URLs used in the Dockerfile are publicly accessible. The administrator should be able to reproduce the Docker image just by running the `docker build` command after decompressing the archive contents in a directory.
2. Prepackaged as a (optionally compressed) tar archive. This can be done in the following way for a Docker image named **publishandsharedemo**

```
docker save -o demo.tar publishandsharedemo:latest
```

The above command will create a tar archive named demo.tar which contains the image. Tar archives can be huge in size. It is generally a good idea to compress it using some format such as zip, before sharing it with the administrator. In Linux like systems, a compressed gzip tar archive can be created in one step as shown below

```
docker save publishandsharedemo:latest | gzip -c > demo.tar.gz
```

The system administrator upon receiving a packaged image can load it in the host machine in the manner shown below. If the image is compressed, it will need to be decompressed first.

```
docker load -i demo.tar


# Use the command below to load an image compressed as a gzip tar archive in
one go
gunzip -c demo.tar.gz | docker load
```

Upon listing the images using `docker images` the `publishandshare:latest` image should be available.

Refer to the 'Developing Tools for the Publish and Share Platform' for instructions on how to register the tool (Docker image) in the database.

Deployment Testing Checklist

After all the components have been setup, use the following table to test if they are working as intended. We will not be checking the validity of the individual services here e.g. we won't check if PublicAPI provides the valid responses. Rather this is more of an integration test to check if the discrete components are working together correctly.

Test	Expectation	Component/Service Involved
Public API availability	• Documentation can be viewed on http://platform.its4land.com/api	• EC2 (main instance)** • Nginx**
Public API - database is connected	• Should return a GeoJSON feature collection for pre-loaded metric map features	• RDS
Geoserver availability	• Admin interface should be reachable on http://platform.its4land.com/geoserver	• EC2 (geoserver instance) • Nginx
S3 read-write access	• POST a contentitem using the PublicAPI (write) - should upload the file and return its UUID	• RDS • S3

Test	Expectation	Component/Service Involved
	• GET a contentitem via its UUID (read) - should download the file	• Cognito • VPC
Process API availability	• Process API is reachable	• EC2 (processapi instance)
Starting tools via Public API	• Start a process using a POST request - process status should change from WAITING → RUNNING / FINSHED	• EC2 (processapi instance) • Docker
Starting interactive tools via Public API	• Start the 'Draw and Make' tool, with the 'Smart Sketch Maps' entrypoint - you should be able to access the tool in the port range (5000-5020) on http://platform.its4land.com:<port number> . The actual port can be viewed from the process logs	• EC2 (processapi instance) • Docker

** *EC2 (main instance)* and *Nginx* are necessary for all test cases

FAQs

1. Why am I unable to view anything on http://platform.its4land.com ?

- Check the EC2 main instance security group settings. Make sure that port 80 and port 443 are accessible externally
- Check if *publishandshare* service is enabled and running on the main instance
    - Check if the service is accessible on *localhost:1080*
- Check if *nginx* service is enabled and running on the main instance
    - Check nginx configuration for the routes

2. Why am I unable to reach geoserver web interface on http://platform.its4land.com/geoserver?

- Check FAQ #1
- Check if geoserver instance is started and running
- Do geoserver instance security group settings allow it to be reachable from the main instance (i.e. it is part of the VPC default security group)?
- Is geoserver service running?
- Is is geoserver accessible from localhost on port 8080?
- Is is accessible from the main instances e.g. 'curl http://172.31.32.200:8080/geoserver'

3. Why I am I unable to access one EC2 instance from another?

- Are both instances running?
- Does the security group configuration allow the instances to be reached from another in the same VPC subnet?

4. Why am I unable to access PublicAPI / Geoserver / ProcessAPI ?

- Are the respective EC2 instances to which they belong started?
- Is the systemd service for the respective application running?
- Are the services accessible from localhost?
- Check FAQ #3

5. Why am I unable to access the RDS database?

- Does the RDS instance exist?

- Has the database been created in the instance?
- Do the security settings allow it to be accessible from the client machine?
- Are the database credentials valid?

6. Why cannot I access S3 objects?

- Can the object be accessed if the Bucket Policy is removed and public access fully allowed?
- Is the VPC endpoint gateway setup correctly?
- Are you using the proper credentials created using AWS Cognito? Do they allow unauthenticated access?
- Does the bucket policy allow access if the request source is changed to allow public access?

7. I cannot access the machine with static IP address?

- Check FAQ #3
- Has the secondary IP address been assigned to the instance in the EC2 management console?
- Do the instances belong to the same subnet? If not the routing needs to be setup properly.
- Check instructions to setup static a IP address.
    - Are the IP address and network interface proper?
    - Did you encounter errors while trying `netplan try`?

8. Why does the EFS filesystem not mount?

- Does the filesystem exist?
- Is the `efs-utils` package installed?
- Do the security settings allow access from within in the VPC?
- Does the mount point exist? Do you have sudo/root permissions to mount it?