ECS Administration Manual

Release 0 TODO:use git id

Latex Author Name

Mar 01, 2017

Contents

1 Choosing the right values for Cores, Memory, Harddisk & Backup 1

2 Install Appliance 3

3 Configure Appliance 5

4 Maintenance 6

5 Digital signed pdf documents 9

6 Development 10

The ecs appliance is a selfservice production setup virtual machine builder and executor. it can be stacked on top of the developer vm, but is independent of it.

Choosing the right values for Cores, Memory, Harddisk & Backup

All data size units in this document are 1024 based (1 GB = 1024 MB, 1 MB = 1024 KB , 1 KB = 1024 bytes)

- Minimum Cpu/Memory (without monitoring): 1 Core, 3GB
- Minimum Cpu/Memory: 2 Cores, 4GB

Core Calculation:

- 20 Studies per Month: 2 Cores
- 40 Studies per Month: 2-4 Cores
- 100-250 Studies per Month: 4-8 Cores

Memory Calculation:

- 2GB Base + 1GB per Core + Disksize of Database (eg. 2GB)
- eg. 2 Cores = 4 GB + 1 GB Database= 5GB
- eg. 8 Cores = 10 GB + 8 GB Database= 18GB

Storage Calculation (10 Years):

- System: 5GB

- Temporary Workspace: 30GB
- Database: (Studies-per-Month/ 12,5)GB
- Document-Storage: (120 * Studies-per-Month * 17)MB

Backup Space Calculation (10 Years):

- Data: 2 (Fullbackups) * max(Document-Storage) + max(Database) + 2 Months delta grow

Limits: The current ecs implementation can sustain 250 Studies per Month for 10 years. Larger implementations need some refactoring in certain parts of the ecs to be efficient.

10 Years Calculation

After 10 Years, the values noted will be reached.

100 Studies per Month

Items	Size	Calculation
Base & Temp	50GB	
Database	8GB	4074220 / 59 * 120
Document-Storage	195GB	100684328 / 59 * 120
Disksize	253GB	52428800 + 8286549 + 204781684
Backup	444GB	2 x 204781684+ 52428800+ 204781684/120*2
recommended Cores	6	
recommended Memory	16GB+	2+ 8+ 6

250 Studies per Month

Items	Size
Base & Temp	50GB
Database	20GB
Document-Storage	500GB
Disksize	570GB
Backup	1332GB
recommended Cores	8
recommended Memory	30GB+

40 Studies per Month

Items	Size
Base & Temp	50GB
Database	3,2GB
Document-Storage	78GB
Disksize	131GB
Backup	178GB
recommended Cores	4
recommended Memory	9,5GB+

20 Studies per Month

Items	Size
Base & Temp	50GB
Database	1,6GB
Document-Storage	40GB
Disksize	92GB
Backup	89GB
recommended Cores	2
recommended Memory	6GB+

Test Instance

Items	Size
Disksize	20-40GB
recommended Cores	1-2
recommended Memory	4GB+

Research data

- EC with ~100 Studies per Month
- Runtime: ~5 Years (59 month) (15.1.2012 - 15.12.2016 ~ 4 Years, 11 Months)
- Studies: 5861 in 59 Months, or ~100 (99,339) Studies per Month

Document Storage

- Current space used: 97GB (100684328KB)
- Directories: 69K Directories
- Files: 296K Files
- Files per Directory: Peak at 4-6 Files per Directory
- average 339KB per File
- average space per study: 17178KB ~ 16,78MB

Postgres Database

- compressed migrated production dump: 475MB
- diskspace used: ~4GB (4074220 KB, 1GB pg_xlog)

Install Appliance

The base of the appliance is Ubuntu Xenial (16.04).

You either need:

- a standard ubuntu cloud image from Ubunu Xenial Cloud Images
  - a cloud-init cidata iso volume with your public key
    - a prebuilt cidata iso with the vagrant user and the insecure vagrant ssh key
      - vagrant-growroot.iso

        * you can build your own iso using github.com/ecs-org/cidata-seed
- an already running empty Ubuntu Xenial and a ssh key to login
  - eg. if your rootserver hoster has a default xenial image
- a local development machine with vagrant and a hypervisor for vagrant installed.
  - vagrant will setup the base machine for you

Partitioning

Unless there is a good reason to do otherwise, the partition layout should be the same as the default xenial cloud & vagrant images layout.

This image consists of a DOS-MBR and partition one taking all the space as the root partition. The Vagrant version has initrd grow-root support, so p1 will resize to maximum on reboot.

custom storage partitions or network attached storage

Add a custom storage:setup object to setup storage partitions. To have seperate volatile and/or data partitions, change storage:ignore:volatile and/or storage:ignore:data to false. Setup will add mountpoints for /data and /volatile. The volatile volume must be labeled "ecs-volatile", the data volume "ecs-data". Use appliance:extra:states and :packages if storage setup needs additional packages installed. See salt/storage/README.md for further information about storage:setup.

install via ssh to a empty xenial vm

ssh into target vm:

```
apt-get -y update; apt-get -y install git
git clone https://github.com/ecs-org/ecs-appliance /app/appliance
cd /
mkdir -p /etc/salt
cp /app/appliance/salt/minion /etc/salt/minion
curl -o /tmp/bootstrap_salt.sh -L https://bootstrap.saltstack.com
chmod +x /tmp/bootstrap_salt.sh
/tmp/bootstrap_salt.sh -X
salt-call state.highstate pillar='{"appliance": {"enabled": true}}'
```

install using vagrant

on your local machine:

```
git clone https://github.com/ecs-org/ecs-appliance ~/ecs-appliance
cd ~/ecs-appliance
vagrant up
```

upgrade a developer vm

Requirement: you need at least 3gb total memory for the appliance.

on a developer vm:

```
# clone from public repository
git clone https://github.com/ecs-org/ecs-appliance /app/appliance
# install saltstack and start appliance install
curl -o /tmp/bootstrap_salt.sh -L https://bootstrap.saltstack.com
```

```
sudo bash -c "mkdir -p /etc/salt; cp /app/appliance/salt/minion /etc/salt/minion; \
    chmod +x /tmp/bootstrap_salt.sh; /tmp/bootstrap_salt.sh -X"
sudo salt-call state.highstate pillar='{"appliance": {"enabled": true}}'
```

if you also want the builder (for building the appliance image) installed:

```
sudo salt-call state.highstate pillar='{"builder": {"enabled": true}, "appliance":
↪{"enabled": true}}'
```

Configure Appliance

Create Config

for a development server, run `cp /app/appliance/salt/pillar/default-env.sls /app/env.yml` and edit settings in /app/env.yml .

for offline environment creation, using your local machine:

- have saltstack installed (minion does not need to run)
- `git clone https://github.com/ecs-org/ecs-appliance ~/path-to-project/ecs-appliance`
- use env-create.sh and env-package.sh like explained below, but add ~/path-to-project/ecs-appliance/salt/common/ to the callpath.
- copy ~/domainname.domain/env.yml to appliance machine at /app/env.yml

```
ssh root@target.vm.ip '/bin/bash -c "mkdir -p /app/"'
scp env.yml root@target.vm.ip:/app/env.yml
```

on an installed but unconfigured appliance

- enter installed but empty appliance
- make a new env.yml: `env-create.sh domainname.domain ~/domainname.domain/`
- edit settings in ~/domainname.domain/env.yml , see comments inside file
- optional, package env into different formats
  - `env-package.sh --requirements; env-package.sh ~/domainname.domain/env.yml`
- print out ~/domainname.domain/env.yml.pdf
- transfer and keep ~/domainname.domain/domainname.env.date_time.tar.gz.gpg

Activate Config

on the target vm:

```
# create a empty ecs database
sudo -u postgres createdb ecs -T template0  -l de_DE.utf8

# activate env and apply new environment settings
chmod 0600 /app/env.yml
cp /app/env.yml /run/active-env.yml
reboot
```

First Internal User setup

After the appliance has rebooted, it configures itself to the new environment. See the progress of the preperation by browsing to https://domainname.domain . After the appliance is ready and shows the login screen, login via ssh to create the first internal office user, with a corresponding client certificate.

```
# create first internal office user (f=female, m=male)
create-internal-user.sh useremail@domain.name "First Name" "Second Name" "f"

# create and send matching client certificate
create-client-cert.sh useremail@domain.name cert_name [daysvalid]

# Communicate certificate transport password over a secure channel
```

Maintenance

Reconfigure a running Appliance

- edit /app/env.yml
- optional, build new env package:
  - first time requisites install, call `env-package.sh --requirements`
  - build new env package call `env-package.sh /app/env.yml`
- activate changes into current environment, call `env-update.sh`
- restart and apply new environment: `systemctl start appliance-update`

Start, Stop & Update Appliance

- Start appliance: `systemctl start appliance`
- Stop appliance: `systemctl stop appliance`
- Update Appliance (appliance and ecs): `systemctl start appliance-update`

Recover from failed state

if the appliance.service enters fail state, it creates a file named "/run/appliance_failed".

After resolving the issue, remove this file using `rm /run/appliance_failed` before running the service again using `systemctl restart appliance`.

Desaster Recovery from backup

- install a new unconfigured appliance as described in chapter install
- copy old saved env.yml to new target machine at /app/env.yml
- reboot new target machine, appliance will configure but stop because of empty database
- ssh into new target machine, execute `recover-from-backup.sh --yes-i-am-sure`

Howto

All snippets expect root.

- enter a running ecs container for most ecs commands it is not important to which instance (web,worker) you connect to, "ecs_ecs.web_1" is used in examples
  - image = ecs, mocca, pdfas, memcached, redis
  - ecs .startcommand = web, worker, beat, smtpd
  - as root `docker exec -it ecs_image[.startcommand]_1 /bin/bash`
    - eg. `docker exec -it ecs_ecs.web_1 /bin/bash`
  - shell as app user with activated environment
    - `docker exec -it ecs_ecs.web_1 /start run /bin/bash`
  - manualy create a celery task:
    - `docker exec -it ecs_ecs.web_1 /start run celery --serializer=pickle -A ecs call ecs.integration.tasks.clearsessions`
  - celery events console
    - `docker exec -it ecs_ecs.web_1 /start run /bin/bash -c "TERM=screen celery -A ecs events"`
  - enter a django shell_plus as app user in a running container
    - `docker exec -it ecs_ecs.web_1 /start run ./manage.py shell_plus`
- make all workflow graphs (with activated environment)

```
./manage.py workflow_dot core.submission | dot -Tpng -osubmission.png
./manage.py workflow_dot notifications.notification | dot -Tpng -onotification.png
./manage.py workflow_dot votes.vote | dot -Tpng -ovote.png
```

- generate ECX-Format Documentation (with activated environment)

```
./manage.py ecx_format -t html -o ecx-format.html
./manage.py ecx_format -t pdf -o ecx-format.pdf
```

- manual run letsencrypt client (do not call as root): `gosu app dehydrated --help`
- destroy and recreate database:

```
gosu app dropdb ecs
gosu postgres createdb ecs -T template0 -l de_DE.utf8
rm /app/etc/tags/last_running_ecs
systemctl restart appliance
```

- get latest dump from backup to /root/ecs.pgdump.gz:
  - `duply /root/.duply/appliance-backup fetch ecs-pgdump/ecs.pgdump.gz /root/ecs.pgdump.gz`
- quick update appliance code:
  - `cd /app/appliance; gosu app git pull; salt-call state.highstate pillar='{"appliance":{"enabled":true}}'; rm /var/www/html/503.html`
- get cummulative cpu,mem,net,disk statistics of container:
  - `docker stats $(docker ps|grep -v "NAMES"|awk '{ print $NF }'|tr "\n" " ")`
- read details of a container in yaml:

  - `docker inspect 1b17069fe3ba | python -c 'import sys,yaml,json; yaml.safe_dump(json.load(sys.stdin),sys.stdout,default_flow_style=False)' | less`

- activate /run/active-env.yml in current shell of appliance vm:
  - `. /usr/local/share/appliance/env.include; ENV_YML=/run/active-env.yml userdata_to_env ecs,appliance`
  - to also set *GIT_SOURCE defaults: `. /usr/local/share/appliance/appliance.include`
- untested:
  - `docker-compose -f /app/etc/ecs/docker-compose.yml run --no-deps ecs.web run ./manage.py shell_plus`
  - most spent time in high.state:
    * `journalctl -u appliance-update | grep -B 5 -E "Duration: [0-9]{3,5}\."`
    * `journalctl -u appliance-update | grep "ID:" -A6 | grep -E "(ID:|Function:|Duration:)" | sed -r "s/.*(ID:|Function:|Duration)(.*)/\1 \2/g" | paste -s -d ' \n' -| sed -r "s/ID: +([^ ]+) Function: +([^ ]+) Duration : ([^ ]+ ms)/\3 \2 \1/g" |sort -n`

Logging

Container:

- all container log to stdout and stderr
- docker has the logs of every container available
  - look at a log stream using eg. `docker logs ecs_ecs.web_1`
- journald will get the container logs via the appliance.service which calls docker-compose
  - this includes backend nginx, uwsgi, beat, worker, smtpd, redis, memcached, pdfas, mocca
  - to follow use `journalctl -u appliance -f`

Host:

- (nearly) all logging is going through journald
- follow whole journal: `journalctl -f`
- only follow service, eg. prepare-appliance: `journalctl -u prepare-appliance -f`
- follow frontend nginx: `journalctl -u nginx -f`
- search for salt-call output: `journalctl $(which salt-call)`

Alerting

if ECS_SETTINGS_SENTRY_DSN and APPLIANCE_SENTRY_DSN is defined, the appliance will report the following items to sentry:

- python exceptions in web, worker, beat, smtpd
- salt-call exceptions and state returns with error states
- systemd service exceptions where appliance-failed is triggered, or appliance_failed, appliance_exit, sentry_entry is called
- internal mails to root, eg. prometheus alerts, smartmond

Metrics

- if APPLIANCE_METRIC_EXPORTER is set, metrics are exported from the subsystems
  - export metrics of: frontend nginx, redis, memcached, uwsgi, cadvisor, process details(uwsgi, postgres, nginx, celery), prometheus node(diskstats, entropy, filefd, filesystem, hwmon, loadavg, mdadm, meminfo, netdev, netstat, stat, textfile, time, uname, vmstat)
  - additional service metrics from: appliance-backup, appliance-update
- if APPLIANCE_METRIC_SERVER is set, these exported metrics are collected and stored by a prometheus server and alerts are issued using email to root using the prometheus alert server
  - there are alerts for: NodeRebootsTooOften, NodeFilesystemFree, NodeMemoryUsageHigh, NodeLoadHigh
  - the prometheus gui is at http://172.17.0.1:9090
  - the prometheus alert gui is at http://172.17.0.1:9093
- if APPLIANCE_METRIC_GUI is set, start a grafana server for displaying the collected metrics
  - grafana is available at http://localhost:3000
- if APPLIANCE_METRIC_PGHERO is set, a pghero instance for postgres inspection
  - pghero is avaiable at http://localhost:5081

Use ssh port forwarding to access these ports, eg. for 172.17.0.1:9090 use "ssh root@hostname -L 9090:172.17.0.1:9090"

Digital signed pdf documents

The ECS uses Mocca and PDF-AS to create digital signed pdf documents.

Mocca and PDF-AS are developed by EGIZ, the eGovernment Innovation Centre of Austria a joint initiative of the Federal Chancellery and the Graz University of Technology, at the Institute for Applied Information Processing and Communications Technology (IAIK).

- EGIZ Mocca - a modular, open source citizen card environment
- EGIZ PDF-AS - a java framework for creating digitial signatures on PDF documents

Mocca

Info:

- Current Mocca: 1.3.23 (2016-04-01)
- Mocca Description
- Mocca Homepage
- bkucommon Configuration
- BKUOnline
- BKUOnline Deployment
- BKUOnline Configuration
- BKUOnline Upgrade Info
- Security Analysis of the Austrian Citizen Card Environment MOCCA and E-Card

Download:

- Download Site

- bkuonline-1.3.23.war

```
sha256=f43f49cbd7ef4df56741097ff5f0637a83cf6cb64701bc484633257ec122dc6a  bkuonline-
↪1.3.23.war
```

- Optional: bkuwebstart-1.3.23.zip

Sourcecode:

- http://git.egiz.gv.at/mocca

PDF-AS

Info:

- Current PDF-AS: 4.0.11 (2016-07-01)
- PDF-AS Description
- Documenation - DE
- Web Interface Documenation - DE
- External Service Documentation - DE
- Profile Documentation - DE
- Goverment Profile Documentation - DE
- PDF-AS Workshop Part 1 - DE
- PDF-AS Workshop Part 2 - DE
- Goverment sponsored Digital Signature Test Site - DE

Download:

- Download Site
- pdf-as-web-4.0.11.war

```
sha256=2008e413032fc926e30b2d666f4363707328a5171a4b170c0fb0599a4e894421  pdf-as-
↪web-4.0.11.war
```

- Optional:
  - pdf-as-web.properties
  - defaultConfig.zip

Sourcecode:

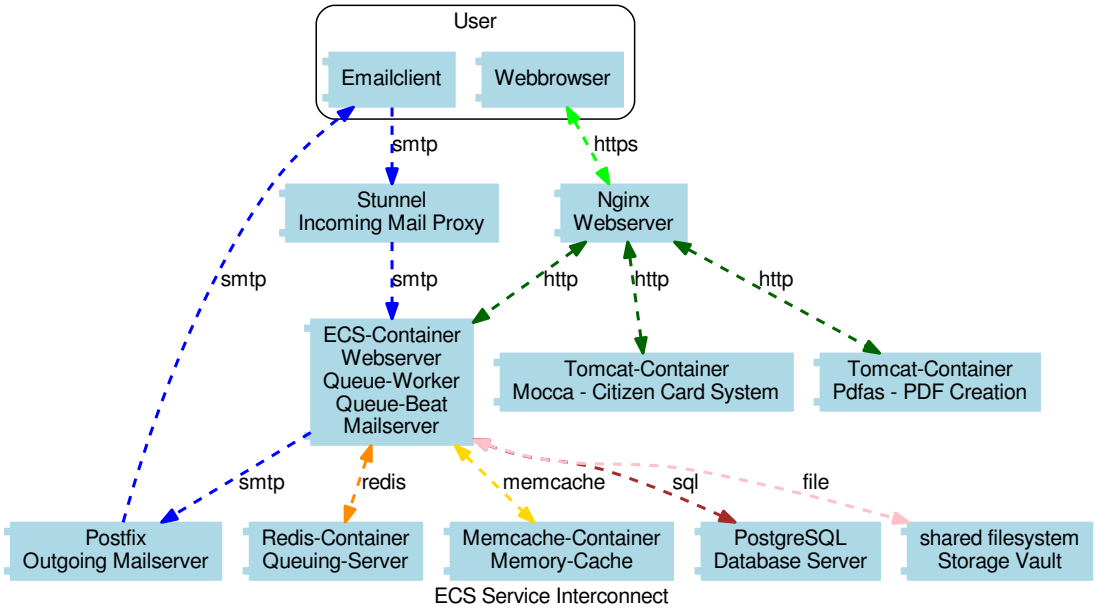
- http://git.egiz.gv.at/pdf-as-4/
- automated builds from source

Development

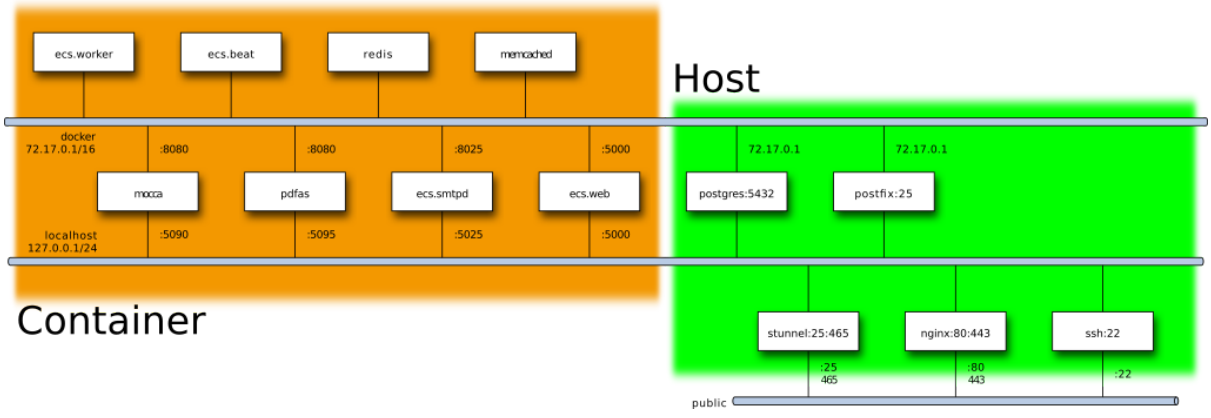
Service Architecture

- Services running on the host system
  - Postgresql Database
  - Stunnel
  - NGINX Webserver

  - SSH Daemon
- Services running inside docker container
  - The ECS Application (Web, Worker, incoming Mail)
  - Redis
  - Memcache
  - tomcat with PDF/AS
  - tomcat with Mocca



- The Appliance uses the default docker network (72.17.0.1/16) for docker container
- Public (Outside) facing Ports
  - NGINX Webserver Ports 80(http) and 443(https)
  - Stunnel Ports 25(smtp) and 465(smtpssl)
  - SSH Daemon 22(ssh)



Repository Layout

Path	Description
/pillar/*.sls	salt environment
/pillar/top.sls	defines the root of the environment tree
/pillar/default-env.sls	fallback env yaml and example localhost ecs config
/salt/*.sls	salt states (to be executed)
/salt/top.sls	defines the root of the state tree
/salt/common/init.sls	common install
/salt/common/env-template.yml	template used to generate a new env.yml
/salt/common/env-create.sh	cli for env generation
/salt/common/env-package.sh	cli for building pdf,iso,tar.gz.gpg out of env
/salt/common/env-update.sh	get env, test conversion and write to /run/active-env.yml
/salt/appliance/init.sls	ecs appliance install
/salt/appliance/scripts/prepare-env.sh	script started first to read environment
/salt/appliance/scripts/prepare-appliance.sh	script started next to setup services
/salt/appliance/scripts/prepare-ecs.sh	script started next to build container
/salt/appliance/scripts/appliance-update.sh	script triggerd from appliance-update.service
/salt/appliance/ecs/docker-compose.yml	main container group definition
/salt/appliance/systemd/appliance.service	systemd appliance service that ties all together

Execution Order

```
[on start]
|
|-- prepare-env
|-- prepare-appliance
|   |
|   |-- optional: call salt-call state.sls appliance.storage.setup
|---|
|
|-- prepare-ecs
|-- appliance
|   |
|   |-- docker-compose up
:
:   (post-start)
|-- appliance-cleanup

[on error]
|
|-- appliance-failed

[on update]
|
|-- appliance-update
|   |
|   |-- apt-daily unattended-upgrades
|   |-- salt-call state.highstate
?   ?-- optional reboot
|   |-- systemctl restart appliance
```

Runtime Layout

Application:

Path	Description
/app/env.yml	local (nocloud) environment configuration
/app/ecs	ecs repository used for container creation
/app/appliance	ecs-appliance repository active on host
/app/etc	runtime configuration (symlink of /data/etc)
/app/etc/tags	runtime tags
/app/etc/flags	runtime flags
/app/ecs-ca	client certificate ca and crl directory (symlink of /data/ecs-ca)
/app/ecs-gpg	storage-vault gpg keys directory (symlink of /data/ecs-gpg)
/app/ecs-cache	temporary storage directory (symlink of /volatile/ecs-cache)
/run/active-env.yml	current activated configuration
/run/appliance-failed	flag that needs to be cleared, before a restart of a failed appliance is possible
/usr/local/share/appliance	scripts from the appliance salt source
/usr/local/[s]bin	user callable programs

Data:

Path	Description
/data	data to keep
/data/ecs-ca	symlink target of /app/ecs-ca
/data/ecs-gpg	symlink target of /app/ecs-gpg
/data/ecs-storage-vault	symlink target of /app/ecs-storage-vault
/data/etc	symlink target of /app/etc
/data/ecs-pgdump	database migration dump and backup dump diretory
/data/postgresql	referenced from moved /var/lib/postgresql

Volatile:

Path	Description
/volatile	data that can get deleted
/volatile/docker	referenced from moved /var/lib/docker
/volatile/ecs-cache	Shared Cache Directory
/volatile/ecs-backup-test	default target directory of unconfigured backup
/volatile/redis	redis container database volume

Container Volume Mapping

Host-Path	Container	Container-Path
/data/ecs-ca	ecs	/app/ecs-ca
/data/ecs-gpg	ecs	/app/ecs-gpg
/data/ecs-storage-vault	ecs	/app/ecs-storage-vault
/volatile/ecs-cache	ecs	/app/ecs-cache
/app/etc/server.cert.pem	pdfas/mocca	/app/import/server.cert.pem:ro

Environment Mapping

Types of environments:

- saltstack get the environment as pillar either from /run/active-env.yml or from a default
- shell-scripts and executed programs from these shellscripts get a flattend yaml representation in the environment (see flatyaml.py) usually restricted to ecs,appliance tree of the yaml file

Buildtime Environment:

- the build time call of `salt-call state.highstate` does not need an environment, but will use /run/active-env.yml if available

Runtime Environment:

- prepare-env

  - get a environment yaml from all local and network sources
  - writes the result to /run/active-env.yml
- appliance-update, prepare-appliance, prepare-ecs, appliance.service
  - parse /run/active-env.yml
  - include defaults from appliance.include (GIT_SOURCE*)
- Storage Setup (`salt-call state.sls appliance.storage.setup`) parses /run/active-env.yml
- appliance-update will call `salt-call state.highstate` which will use /run/active-env.yml
- appliance.service calls docker-compose up with active env from /run/active-env.yml
  - docker compose passes the following to the ecs/ecs* container
    * service_urls.env, database_url.env
    * ECS_SETTINGS
  - docker compose passes the following to the mocca and pdfas container
    * APPLIANCE_DOMAIN as HOSTNAME