ECS Administration Manual

Release 0 TODO:use git id

Latex Author Name

Mar 10, 2017

Contents

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.

Hosting Requirements

The ecs software is designed to run as an external available internet webservice, reachable by everyone.

Requirements

- a Virtual Machine on a supported hypervisor or Hardware sized according to the *Assessment* Chapter
- a Backup space (for encrypted backup data) accessable by one of 25 supported storage protocols explained under *Backup Storage*
- incoming ports 22,25,80,443,465 of a fixed public IPV4 address and DNS, IPV4 setup as described under *Internet Connectivity*

Virtual Machine

- The base of the virtual machine is a Ubuntu Xenial (16.04 LTS) 64Bit Standard Cloud Image.
- The appliance was tested on the following hypervisors: xen,kvm,vmware sphere,virtualbox
  - hyperv is not tested but is expected to work

  - rollouts to amazon ec2, google gce or openstack clouds are not tested but meta data gathering from ec2, gce or openstack compatible meta data services is implemented so it should work but probably need some tweaking beforehand

- Follow the Assessment Chapter for the right sizing of cpu-cores, memory and harddisk.

Backup Storage

- For storing backup data the appliance needs a storage space accessable via one of the 24 duplicity supported storage protocols or via cifs.
- In addition to the supported duplicity protocols the appliance has support for **cifs** (windows file sharing attached volumes, including automatic mount and unmount)
- Tested protocols so far: localfile, ftp, ftpssl, ssh/scp, ssh/sftp, http/webdav, cifs
- For Details see: Duplicity Manual - Section URL-Format
- Storage at Rest: All backup data is encrypted before it leaves the machine using gpg (GnuPG) and is saved without any prior decryption on the target space.
- The storage space may be hosted internal or external, using the same provider as the machine hosting or using a different provider. The hosting provider may be a trusted or untrusted third-party.
- Rotation and Retention is automatic, the backup process is unattended.
- See *Assessment* for the correct storage space size.

Internet Connectivity

- permanent internet connectivity with a strong (>20Mbit) upload channel
- an ip address and dns server settings served to the machine by DHCP for automatic configuration
  - this can be a internal or the public IPv4 Address if the hosting location needs this, but it must be served by DHCP
  - Unusable internal nets are 172.17.X.X and 10.0.3.X, because these are used by the appliance itself
- a dns [sub]domain with a A and a MX Entry and the reverse ptr of the public IPv4-Adress set to the domain name
  - a A Record pointing to the public IP
  - a MX Record pointing to the A Record
  - a reverse PTR Record pointing to the domain name

Examples for domains “https://whatever.me” and “https://another.sub.domain.me”:

```
whatever.me. IN A 1.2.3.4
whatever.me. IN MX 10 whatever.me
4.3.2.1.in-addr.arpa. IN PTR whatever.me.

another.sub.domain.me. IN A 5.6.7.8
another.sub.domain.me. IN MX 10 another.sub.domain.me
8.7.6.5.in-addr.arpa.  IN PTR another.sub.domain.me
```

- a public IPv4-Address or the incoming ports 22,25,80,443,465 of this address forwarded to the machine
  - port 22 - ssh: ssh publickey auth is used. this connection is used for installation and optional support
  - port 25,465 - smtp: the incoming email server of the appliance will receive emails from answers of forwarded ecs communication. if the hosting locations policy does not permit this, the ecs will work without it, but loses email answering functionality

  - port 80 - http: the appliance communicates only over https, but needs http for letsencrypt client certificate renewal and for redirecting to the https site
  - port 443 - https: the appliance uses letsencrypt to issue a host certificate and lets internal user administer https client certificates in selfservice

Certificates

The appliance uses LetsEncrypt to issue https/ssl host certificates and also takes care of the renewal of these host certificates. Https client certificates are issued by the appliance itself and can be done in selfservice as an internal ecs webfrontend user. There is no IT-administration task needed in this process.

"Firewall"/Endpointprotection/"Antivirus" Security Products:

The appliance does not need a firewall but works as long as the incoming ports listed are forwarded to the machine and outgoing traffic from the machine is permitted.

Warning: many security products are known to disturb/break HTTPS Host Certificate and Client Certificate validation and weaken the transport protocols on the wire. See The Security Impact of HTTPS Interception .

The ecs appliance uses https client certificates for protection of elevated user rights. If the hosting location has some mandatory security product, you probably need some extra configuration in the security product to fix support for Client Certificates.

This may also apply to to the pc's of internal ecs desktops. Some desktop "antivirus" products may need similar extra configuration to have working https client certificate support. So far there have been findings of Kaspersky and McAfee products to need extra configuration.

Cores, Memory, Harddisk & Backup Space Assessment

All data size units in this document are 1024 based.

- 1 GB = 1024 MB, 1 MB = 1024 KB , 1 KB = 1024 bytes

As a guideline, if you have more memory available, add some extra memory.

Warning: If you deviate from the example values below, be sure to add the needed memory per core if you add more cores, because many of the appliance processes get scaled to the number of cores.

- 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 are possible with some refactoring in certain parts of the ecs.

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)

Deploy Appliance

The base of the appliance is Ubuntu Xenial (16.04 LTS). Installation is done unattended using ssh for login to the machine.

as a virtual machine

You either need:

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

        - vagrant-publickey-growroot.iso
        - vagrant-password-growroot.iso
      * you can build your own seed 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

Start the new virtual machine and login via ssh.

on hardware or other custom configurations

Needed:

- an already running empty Ubuntu Xenial and a ssh key to login
  - eg. if your rootserver hoster has a default xenial image

In a typical root server hosting setup there are two harddisks per machine. These should be configured as a raid1 (mirroring) setup with lvm on top of it.

Example:

- Hardware: Hetzner px61nvme Rootserver
- hetzner setup config

```
DRIVE1 /dev/nvme1n1
DRIVE2 /dev/nvme0n1
SWRAID 1
SWRAIDLEVEL 1
BOOTLOADER grub
HOSTNAME Ubuntu-1604-xenial-64-minimal
PART /boot ext3 512M
PART lvm vg0 all
LV vg0 root / ext4 300G
IMAGE /root/.oldroot/nfs/install/../images/Ubuntu-1604-xenial-64-minimal.tar.gz
```

Network attached storage and custom partitioning

For a virtual machine deployment it is best to stick to the default xenial cloud images layout, unless there is good reason to deviate.

The appliance supports two possible external network attached storage volumes, one for permanent data and one for volatile data. To have seperate volatile and/or data partitions, change storage:ignore:volatile and/or storage:ignore:data to false. The volatile volume must be labeled “ecs-volatile”, the data volume “ecs-data”. Setup will add mountpoints for /data and /volatile. Use appliance:extra:states and :packages if storage setup needs additional packages installed.

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, 4gb minimum if you want metrics active.

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

The Appliance is configured using a yaml confiuration file which can be placed in one of the following locations:

- absolute filepath set via environment variable ENV_YML
- local file at `/app/env.yml`
- a attached drive with label cidata (cloud-init: no-cloud config)
- a attached drive with label config-2 (cloud-init: openstack config)
- aws-ec2 (amazon elastic computing cloud) meta-data server
- gce (google compute engine) meta-data server

It will be copied to `/run/active-env.yml` (ramdisk) on each appliance startup.

:warning: Do **not** reuse an already existing configuration for a different domain/instance, **always** create a new configuration, to not leak secrets between domains.

Create a new Configuration

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`
- transfer, print out ~/domainname.domain/env.yml.pdf and store in a safe place.
- save a encrypted copy of env.yml in safe place.
- **Important**: The env.yml contains all needed secrets for a working appliance and is also the one needed piece of information if you want to recover from backup in case of a storage failure.

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
```

for a development server:

- run `cp /app/appliance/salt/pillar/default-env.sls /app/env.yml` and edit settings in "/app/env.yml".

Modify Configuration

Any applied change in the config file will reconfigure the appliance on the next appliance-update run to the new values found in the configuration.

Eg. if you want to change your backup target to a different location, just change the value and restart the appliance, it will detect and configure all needed changes to the environment.

See the comments in the configuration for different possibilities for the appliance configuration.

```
#cloud-config
# XXX keep the "#cloud-config" line first and unchanged, software expects this␣
↪header

ssh_authorized_keys:
  # # you can put your ssh keys here, this is also used by cloud-init
  # - "ssh-rsa and some long glibberish somebody@somewhere"
ssh_deprecated_keys:
  # # you can copy deprecated keys here,
  # # state.highstate will remove these keys from allowed login,
  # # additionaly this section serves as log of past access keys
  # - "ssh-rsa and some long glibberish somebody@somewhere"
disable_root: false
# disable_root set to false for cloud-init compatibility, appliance expects root␣
↪to be usable

appliance:
  # # standby: default false, if set appliance will not activate
  # standby: true
  domain: {{ domain }}
  allowed_hosts: {{ domain }}
```

```
  ssl:
    letsencrypt:
      enabled: true
    # # client_certs_mandatory, default false, if true, always need a client␣
↪certificate
    # client_certs_mandatory: true
    client_certs_mandatory: false
    # # key: if set ssl key for https host will be used, default empty
    # key: filename-key.pem
    # # cert: if set ssl key for https host will be used, default empty
    # cert: filename-cert.pem
  # # sentry:dsn set to your sentry url, may be the same as ecs:settings:SENTRY_DSN
  # sentry:
  #   dsn: 'https://url'
  # metric:
  #   exporter: false
  #   server: false
  #   gui: false
  #   pghero: false
  # git:
  #   # default see appliance.include
  #   branch: master
  #   source: git_url
  # extra:  # write out extra files on appliance configure
  #   files:
  #     - path: /path/of/filename
  #       content: |
  #           # Your content here
  #       owner: user:group
  #       permissions: "0600"
  #   states: # include extra states at state.highstate
  #     - qrcode
  #   packages: # include extra packages at state.highstate
  #     - ghostscript
  storage:
    # # setup: optional, will be executed by appliance.storage.setup if volatile␣
↪or data can not be found
    # setup: |
    # proxy_cache: true
    # # default false, if true 10 additional GB diskspace are used
    # # for operating polipo, a http proxy cache
    ignore: # default false, if true: will not look for ecs-volatile or ecs-data␣
↪filesystem
      volatile: true
      data: true
  backup:
    url: file:///volatile/ecs-backup-test/
    # options: "string of options directly passed to duplicity"
    # # mount default empty, script will mount & unmount source to target on␣
↪backup run
    # mount:
    #   type: "cifs"
    #   source: "//1.2.3.4/datavolume"
    #   target: "/mnt/appliance-backup-mount"
    #   options: "user=username,pass=password"
    # # options are passed to mount via "-o"
    encrypt: |
        a ascii armored GPG Key
ecs:
  # git: # default see appliance.include
  #   branch: stable
  #   source: git_url
  userswitcher:
```

```
  enabled: false
  # set userswitcher to enabled on a test instance to change to different users
settings: |
    DOMAIN = '{{ domain }}'
    ABSOLUTE_URL_PREFIX = 'https://{}'.format(DOMAIN)
    ALLOWED_HOSTS = [DOMAIN, ]
    PDFAS_SERVICE = ABSOLUTE_URL_PREFIX+ '/pdf-as-web/'
    # use PDFAS_SERVICE = "mock:" for mock signing
    SECURE_PROXY_SSL = True
    ECS_REQUIRE_CLIENT_CERTS = True
    DEBUG = False
    # SENTRY_DSN = 'https://url' # set to sentry url if available
    ETHICS_COMMISSION_UUID = 'ecececececececececececececececec'
    # set ETHICS_COMMISSION_UUID to the desired UUID of the target commission

    SECRET_KEY = '{{ ssl_secret() }}'
    REGISTRATION_SECRET = '{{ ssl_secret() }}'
    PASSWORD_RESET_SECRET = '{{ ssl_secret() }}'

    EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
    EMAIL_BACKEND_UNFILTERED = 'django.core.mail.backends.smtp.EmailBackend'
    EMAIL_UNFILTERED_DOMAINS = () # = ('example.com',)

    SMTPD_CONFIG['listen_addr'] = ('0.0.0.0', 8025)
    SMTPD_CONFIG['domain'] = DOMAIN

    # User registration, password reset, send client certificate and mail to
    # receivers at a domain included in EMAIL_UNFILTERED_DOMAINS will be sent via
    # EMAIL_BACKEND_UNFILTERED. All other mail will be sent via EMAIL_BACKEND.
    #
    # Backend to use to NOT sent email but log email to console:
    #   django.core.mail.backends.console.EmailBackend
    #
    # Backend to use to send via EMAIL_* smtp settings:
    #   django.core.mail.backends.smtp.EmailBackend
    #

vault_encrypt: |
    a ascii armored GPG Key

vault_sign: |
    a ascii armored GPG Key
```

Ethics Commission UUID's

23d805c6b5f14d8b9196a12005fd296	Ethikkommission der Medizinischen Universität Wien
7b51f38bde8a4161a0dc34647fc7e65	Ethikkommission Krankenhaus Barmh.Brüder - Wien
85dc386061584fbe8549ce4e4d828fb	Ethikkommission der Stadt Wien gemäß KAG, AMG und MPG
d6a22c635a584521b107481ac1831816	Ethikkommission Forschungsinstitut des Wiener Roten Kreuzes
55ae93ec9df04d6abfc8d233ec5ccf8e	Ethikkommission Krankenhaus Barmh.Schwestern - Wien
7cd6d52120b3474ba502931b9f60a5f	Ethikkommission Confraternität-Priv.Klin. Josefstadt und Priv.Klin. Döbling
7df9ebaf15434709b09c3def9a6c8769	Ethikkommission St.Anna Kinderspital
f122f144616541d391fde2dcc761aff4	Ethikkommission Österreichische Arbeitsgemeinschaft für Klinische Pharmakologie
25b6744780434a3f96a1e43b405d384	Ethikkommission Privatkrankenanstalt Rudolfinerhaus
d542994ced34403db841786a1c1ab89	Ethikkommission Rheuma-Zentrum Wien-Oberlaa
5615df-baf8c8445d960d1e2cd9c00dc3	Ethikkommission Krankenhaus des Göttlichen Heilandes
4d3a2d5f138940f293ee87fe6ec1d5b2	Ethikkommission Evangelisches Krankenhaus
8d2950e3a0294f68bde647a54df6d82	Ethikkommission der Allgemeinen Unfallversicherungsanstalt
9f6b509e716e413f865d95bdd630e9b	Ethikkommission Immunologische Tagesklinik
b17f32f604fa4452b5ff3a2baa9e0704	Ethikkommission des Landes Niederösterreich
6688ce16a3b84d42b1531389e603989	Ethikkommission Krankenhaus Elisabethinen
e4dcd05a31ad475ca72dea7b84ef030	Ethikkommission des Landes Oberösterreich
e269491bb9c040aaad6a5f11df343f3	Ethikkommission Krankenhaus Barmh.Brüder - Linz
1cca34032077445d95dabf7802fade2	Ethikkommission Krankenhaus Barmh.Schwestern - Linz
39cbb589ef044d27bceb6ee5ac796ae	Ethikkommission für das Bundesland Salzburg
280414583b894c809a9baa8134d7fe4	Ethikkommission der Medizinischen Universität Innsbruck
183881da8200493aa7edd8bebeea75b	Ethikkommission des Landes Vorarlberg
95821eba88f34b2195f96e747d7f6b1	Ethikkommission des Landes Burgenland gemäß KAG, AMG und MPG
6e7cfab5f8cd40df83c9de4fac9bb20f	Ethikkommission des Landes Steiermark gemäß AMG und MPG
75b5a9714f354a5b842aa0102914803	Ethikkommission Krankenhaus Barmh.Brüder - Graz
9287fb0878b94f9f9d90b5582d1cefca	Ethikkommission Krankenhaus Barmh.Brüder - Eggenberg
e17ee744e4e840778f316cf3f79de6b4	Ethikkommission der Medizinischen Universität Graz
ec9f179a52a04a7aa52446232c3fd4bd	Ethikkommission des Landes Kärnten
c890205dcb7543c8a76bf324512c5f81	Ethikkommission des Krankenhaus St. Josef
dc1b115d9809461ba3ea9450b079ddd	Kommission für Scientific Integrity und Ethik der Karl Landsteiner Privatuniversität

Activate Configuration

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`.

Howto (Admin Snippets)

commands in 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 as example.

- image = ecs, mocca, pdfas, memcached, redis
- ecs.startcommand = web, worker, beat, smtpd
- as root `docker exec -it ecs_image[.startcommand]_1 /path/to/command`
  - 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

```
docker exec -it ecs_ecs.web_1 /start run /bin/bash
./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

```
docker exec -it ecs_ecs.web_1 /start run /bin/bash
./manage.py ecx_format -t html -o ecx-format.html
./manage.py ecx_format -t pdf -o ecx-format.pdf
```

commands for the appliance host

All snippets expect root.

- 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`
- 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`

Backup Configuration

- Backup Target can be a (untrusted) third third party
- backup is done using duplicity
- Cycle: Backup is done once per day around 00:30
- Contents: It consists of a fresh database dump and the contents of /data/ecs-storage-vault
- Type, Rotation & Retention:
  - Backup will start with a full backup and do incremental backups afterwards
  - 2 Months after the first full backup a second full backup will be created
  - Rotation: Every Backup (full or incremental) will be purged after 4 Months

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`

Automatic Updates

Updates are scheduled depending appliance:update:schedule once per day or once per week on sunday around 06:30.

Depending the types of updates available the update will take between 1 and 5 minutes in most cases, 5-10 minutes if the ecs container will be rebuild and up to 30 minutes if there are database migrations to be executed.

The following items are updated:

- system packages are updated, including a reboot if the update need a kernel reboot
- lets encrypt certificatges are updated
- appliance source will be updated and executed
- ecs source will be updated and executed
  - the ecs-docs source will be updated

Logging Configuration

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:

- all logging (except postgres) 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 Setup

if ECS_SETTINGS_SENTRY_DSN and APPLIANCE_SENTRY_DSN are 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 or service-failed is triggered
- shell calls to appliance.include: appliance_failed, appliance_exit, sentry_entry
- internal mails to root, eg. prometheus alerts, smartmond

Metrics Collection

- 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

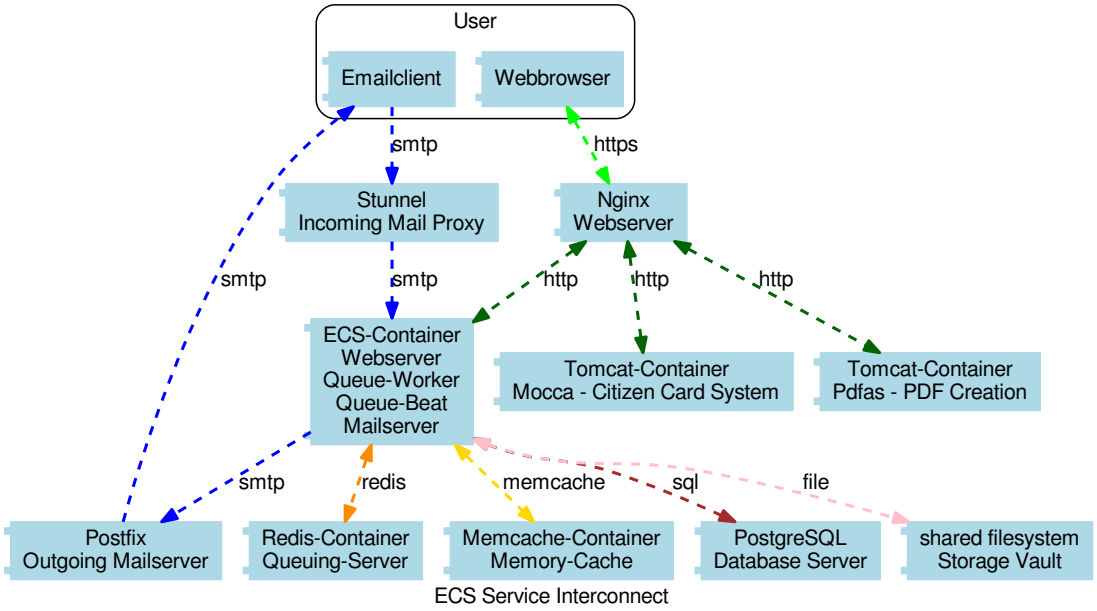
Sourcecode:

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

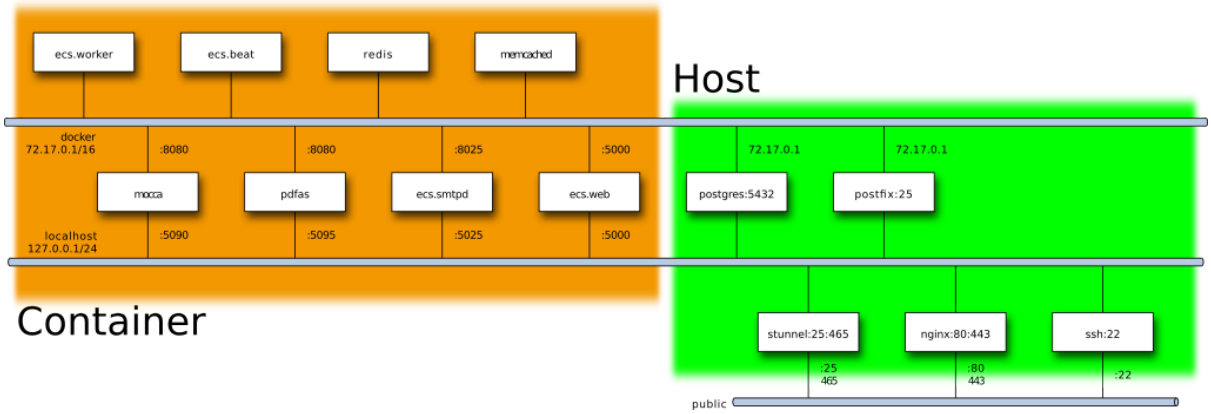
Development

Service Architecture

- Ubuntu Xenial (16.04 LTS) 64Bit Standard Cloud Image as Base
- Services running on the host system
  - Webserver: NGINX 1.10
  - Database: Postgresql 9.5 Database
  - SMTP/SSL Proxy: Stunnel
  - SSH Daemon: openssh
- Services running inside docker container
  - The ECS Application (Web, Worker, incoming Mail)
    - Django is served via uwsgi inside the web container
    - Celery is used for asyncronus worker & beat
    - python smtpd is used for incoming Mail processing
  - Redis for Queuing Services
  - Memcache for Caching Services
  - tomcat running PDF/AS for Eletronical Signed PDF Generation
  - tomcat running Mocca for accessing the Digital ID-Card used for signed PDF-Generation



- 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