Aeternity node architecture

Ulf Wiger
Dimitar Ivanov

Structure and build

- Non-distributed Erlang node
- rebar3 build strategies
- Common Test suites
- Docker build support
  - Used not least in system_test

```
<root>
  config
  apps
      aecore
      aeutils
      aevm
      ...
  system_test
  _build
```

Extracted apps:
- aeminer
- aebytecode
- aeserialization
- aestratum
- ecrecover
- enoise
- mnesia_rocksdb

The ’setup’ application

- Controls location of data and log files
- Dynamic expansion of application env vars
- Fine control of system initialization order (next slide)

From config/sys.config:

```
{setup, [
         {abort_on_error, true},
         {data_dir, "data"},
         {log_dir, "log"}
        ]}
```

From aecore/aecore.app.src:

```
  {env, [
         {exometer_predefined,
          {script,
"$PRIV_DIR/exometer_predefined.script"}},
         {exometer_defaults,
          {script,
"$PRIV_DIR/exometer_defaults.script"}},
```

https://github.com/uwiger/setup

System start order

- The ’setup’ application runs initialization hooks when started
  - The hooks are locally defined in each .app.src file
- Numbered MFA hooks executed synchronously, in numeric order
  - Order within a numbered ’phase’ is undefined

From aeutils/aeutils.app.src:

```
{'$setup_hooks',
 [
  {normal, [
             {100, {aeu_env, read_config, []}}
            ]}
 ]}
```

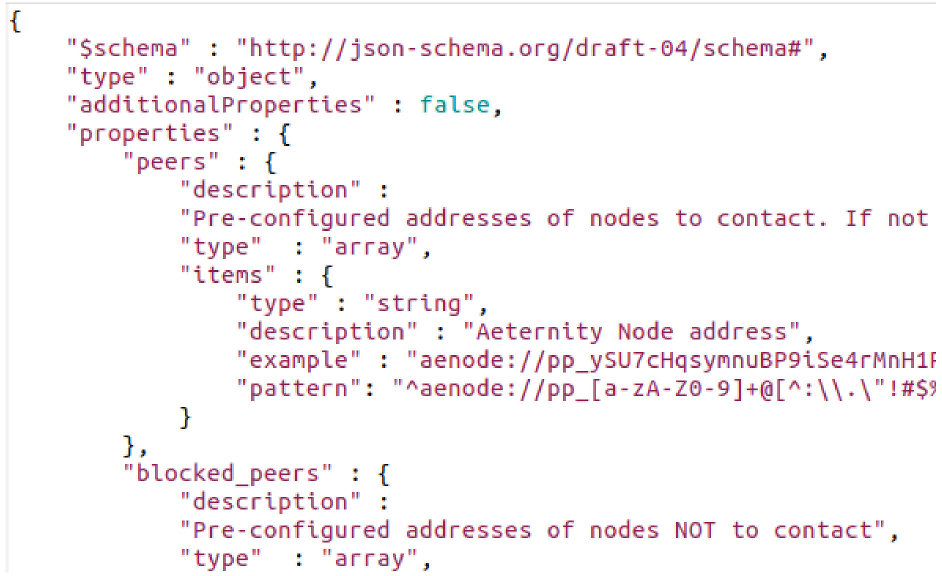
From aecore/aecore.app.src:

```
{'$setup_hooks',
 [
  {normal, [
             {110, {aecore_app, check_env, []}},
             {110, {aec_dev_reward, ensure_env, []}},
             {110, {aehttp_app, check_env, []}},
             {110, {aec_hard_forks, ensure_env, []}},
             {110, {aec_mining, check_env, []}},
             {200, {aec_db, check_db, []}}
            ]}
 ]}
```

Configuration and environment

- aeu_env:read_config() (phase 100) reads the .[yaml|json] config
- The check_env() functions in phase 110 read and cache/optimize env/config
- The aeu_env API has evolved over time
  - Lots of legacy env var handling around

From aeutils/priv/aeternity_config_schema.json

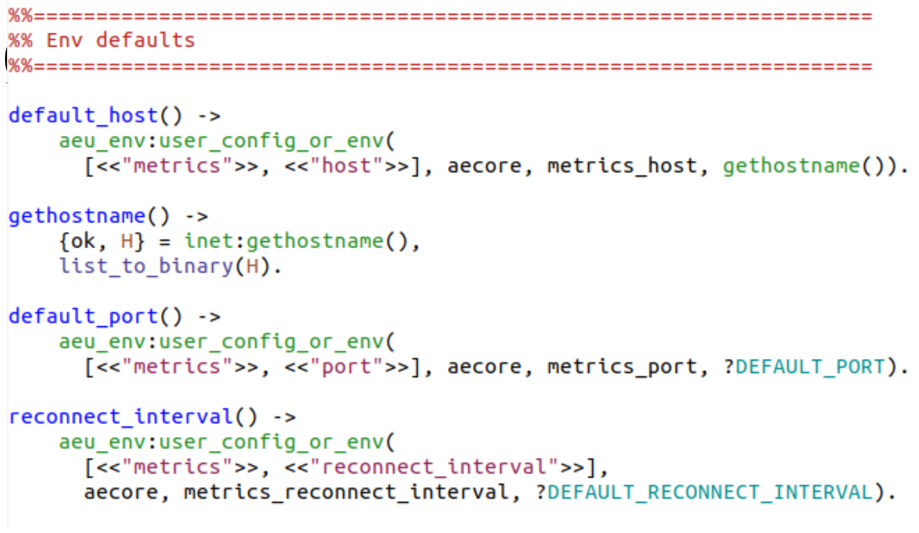


```
{
    "$schema" : "http://json-schema.org/draft-04/schema#",
    "type" : "object",
    "additionalProperties" : false,
    "properties" : {
        "peers" : {
            "description" :
            "Pre-configured addresses of nodes to contact. If not
            "type"  : "array",
            "items" : {
                "type" : "string",
                "description" : "Aeternity Node address",
                "example" : "aenode://pp_ySU7cHqsymnuBP9iSe4rMnH1
                "pattern": "^aenode://pp_[a-zA-Z0-9]+@[^:\\.\"!#$
            }
        },
        "blocked_peers" : {
            "description" :
            "Pre-configured addresses of nodes NOT to contact",
            "type"  : "array",
```

Example of 'evolved' env checking

- `aeu_env:user_config_or_env()`
  - First checks the user confi
  - Then app environment
  - Then hard-coded default

From aecore/src/aec_metrics.erl



```
%%===================================================================
%% Env defaults
%%===================================================================

default_host() ->
    aeu_env:user_config_or_env(
      [<<"metrics">>, <<"host">>], aecore, metrics_host, gethostname()).

gethostname() ->
    {ok, H} = inet:gethostname(),
    list_to_binary(H).

default_port() ->
    aeu_env:user_config_or_env(
      [<<"metrics">>, <<"port">>], aecore, metrics_port, ?DEFAULT_PORT).

reconnect_interval() ->
    aeu_env:user_config_or_env(
      [<<"metrics">>, <<"reconnect_interval">>],
      aecore, metrics_reconnect_interval, ?DEFAULT_RECONNECT_INTERVAL).
```

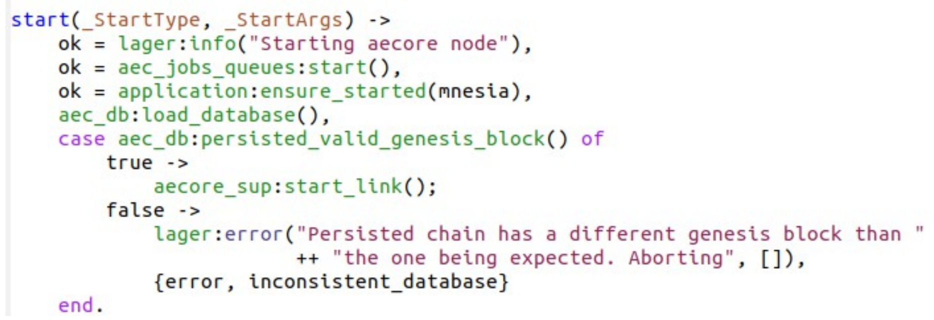
JSON-Schema Syntax Check

- The `jsx` parser doesn’t give informative error info
- Suggestion: If schema fails to compile, paste contents int something like https://www.jsonschemavalidator.net/
- The aeternity `check_config <config>` command will run a schema validation (no helpful syntax check)

Database initialization

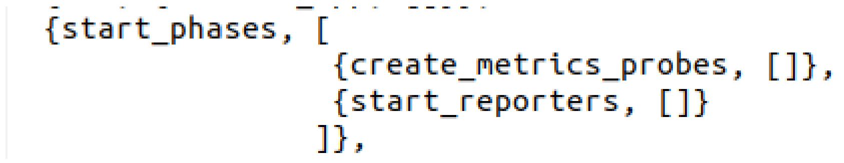
- Mnesia is listed as load-only in the relx `'release'`
- Started explicitly from `aec_db`, called from `aecore_app:start/2`
- `Aec_db:check_db()` was previously called from start phase 200
  - Prepares database backend
  - Checks existing db
  - Possibly creates empty db
- The `mnesia_rocksdb` backend is maintained by Aeternity

From aecore/src/aecore_app.erl



```erlang
start(_StartType, _StartArgs) ->
    ok = lager:info("Starting aecore node"),
    ok = aec_jobs_queues:start(),
    ok = application:ensure_started(mnesia),
    aec_db:load_database(),
    case aec_db:persisted_valid_genesis_block() of
        true ->
            aecore_sup:start_link();
        false ->
            lager:error("Persisted chain has a different genesis block than "
                        ++ "the one being expected. Aborting", []),
            {error, inconsistent_database}
    end.
```

From aecore/src/aecore.app.src

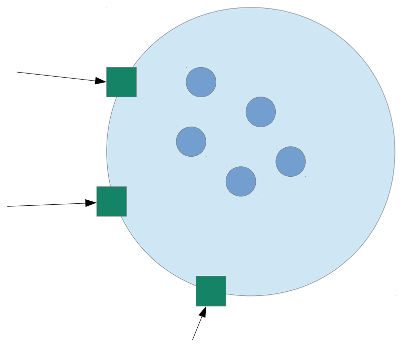


```erlang
  {start_phases, [
                  {create_metrics_probes, []},
                  {start_reporters, []}
                 ]},
```

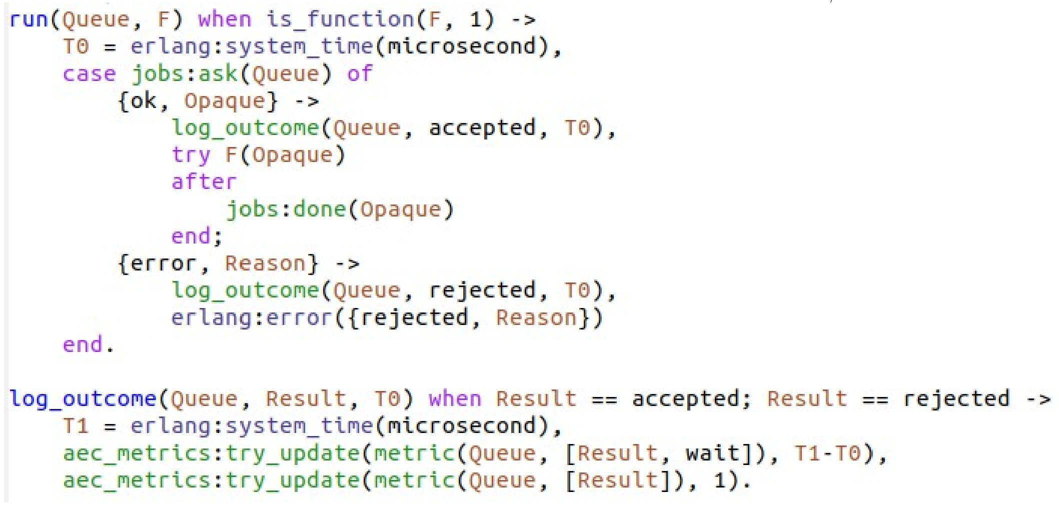
(More on metrics in other presentation)

Jobs: Load regulation

- Principle: Regulate load at the edges of the system
- Jobs puts requests in designated queues
  - Then pulls jobs at configured batch size/frequency
- Some queues can be tweaked in user config
- Queue API: `aec_jobs_queues.erl`



From aecore/src/aecore_app.erl



```
run(Queue, F) when is_function(F, 1) ->
    T0 = erlang:system_time(microsecond),
    case jobs:ask(Queue) of
        {ok, Opaque} ->
            log_outcome(Queue, accepted, T0),
            try F(Opaque)
            after
                jobs:done(Opaque)
            end;
        {error, Reason} ->
            log_outcome(Queue, rejected, T0),
            erlang:error({rejected, Reason})
    end.

log_outcome(Queue, Result, T0) when Result == accepted; Result == rejected ->
    T1 = erlang:system_time(microsecond),
    aec_metrics:try_update(metric(Queue, [Result, wait]), T1-T0),
    aec_metrics:try_update(metric(Queue, [Result]), 1).
```