MAESTRO – Orchestrating Large Scale Multiplayer Games

pedro.silva@miniclip.com / pedro.engana@miniclip.com

Erlang User Conference 2017

About us…

- **Pedro Silva - Backend Development Lead @ Miniclip**
  - 18 years software engineering from embedded to distributed systems and game servers development
  - open source contributor and enthusiast (@github - posilva)
  - passionate about Erlang (since 2014)

- **Pedro Engana - Backend Software Engineer @ Miniclip**
  - 5 years doing a bit of everything in game development: from physics engines, 3d rendering, UI, artificial intelligence and currently game servers

Agenda

- About Miniclip
- Backend @ Miniclip
- What is Maestro
- The Fundamentals
- Deployment and Monitoring
- The Maestro Ecosystem
- Trade-offs and Drawbacks

MINICLIP PLAY GAMES

THE HISTORY OF MINICLIP | 2001-2010

Becoming Web leaders in Online Game Distribution

2001

Miniclip.com launches

2001-02: Pioneers game syndication model

2004

Google

Miniclip is No.1 in Online Games Google ranking

RuneScape

First 3rd party MMO, quickly becomes the world's biggest MMO

2005

Unilever

Huge in-game marketing campaign marks new advert games service

THE WEBBY AWARDS

2006

CLUB PENGUIN

Club Penguin is marketed and distributed on Miniclip.com

THE WEBBY AWARDS

2007

Over 500 games published on Miniclip.com

THE WEBBY AWARDS

The first and only company to win the Webby "Best Game"site for three years

2008

THE SUNDAY TIMES TECH TRACK 100

Ranked 26th in the "Times Tech Track 100" league table

Rob Small, Miniclip CEO, named London Entrepeneur of the Year by E&Y

2009

Monster Trucks Nitro

First iOS game franchise launched "Monster Trucks Nitro"

THE WEBBY AWARDS

2010

Lisbon mobile studio established

FRAGGER

Fragger launches at No.1 and No.3 on UK and US overall paid iOS app charts

MINICLIP PLAY GAMES

THE HISTORY OF MINICLIP | 2011-2017

How we evolved from Web to a mobile Top Grossing Company

MINICLIP
PLAY GAMES

GLOBAL AUDIENCE - 196 COUNTRIES

60+ TITLES RELEASED | OVER 10+ AWARDS

More than 800M downloads

183M MAU

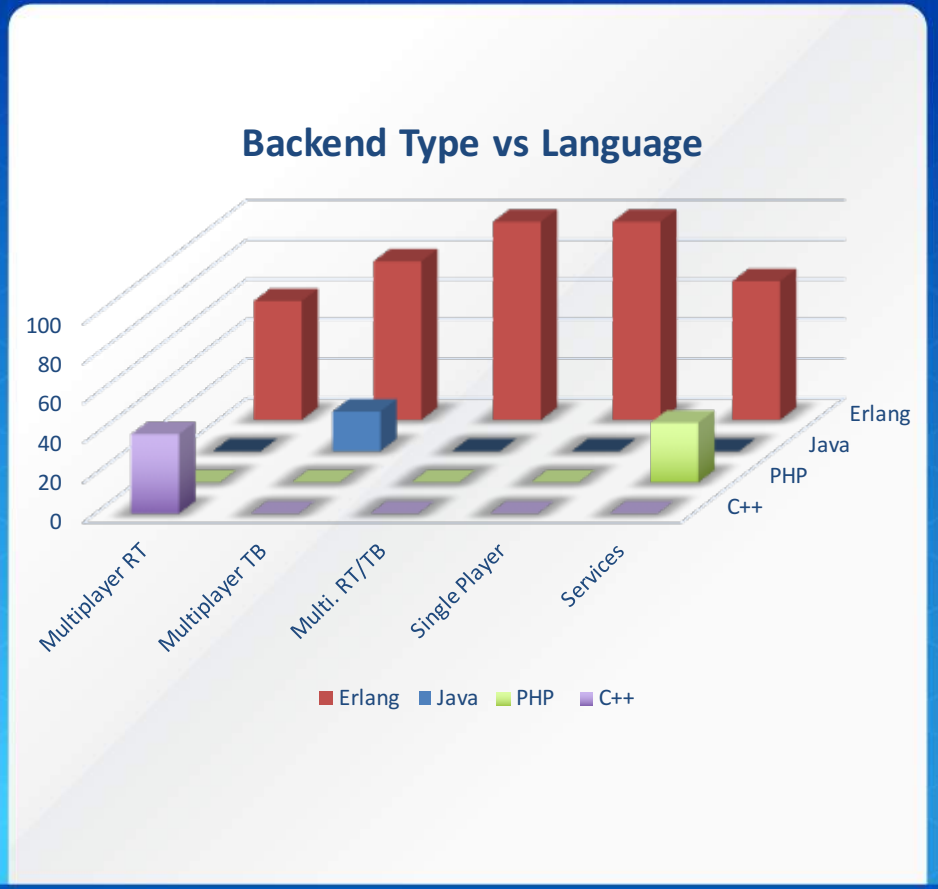
800+K PCU

25M DAU


BACKEND DEVELOPMENT?

Backend @ Miniclip

- We use Erlang in all types of backend services/servers
- C++ is used in game servers with realtime requirements
- All our backend engineers have C++ development background
- We have PHP and Java in some services but we are porting to Erlang, whenever it makes sense



Backend @ Miniclip

Services

- Video Ads rewards
- Single player games RESTFul API servers
- Social Features (HTTP/ workers)
- Analytics

Single-Region Clusters

- Turn-Based Multiplayer Games (PvP)
- Massive persistent connections (10's of nodes, ~35k users per node)
- Non latency sensitive game

Multi-Region Clusters

- Soft realtime games (PvP), latency sensitive
- Arena mode (.IO) games (game server engine)
- 250 - 5k users per node

What is Maestro… (some context)

Early 2016 - New mobile .IO game

Stakeholders requirements:

- Time to market (3 months to release world wide)
- Small team (2 client + 1 backend developer)
- Implement mobile(.IO) base features (MVP)

Technical challenge (initial):

- Game was develo
- Game s e) and client in C++ + C col, using web socket
- Implement in C++ all base features that we already have implemented in Erlang

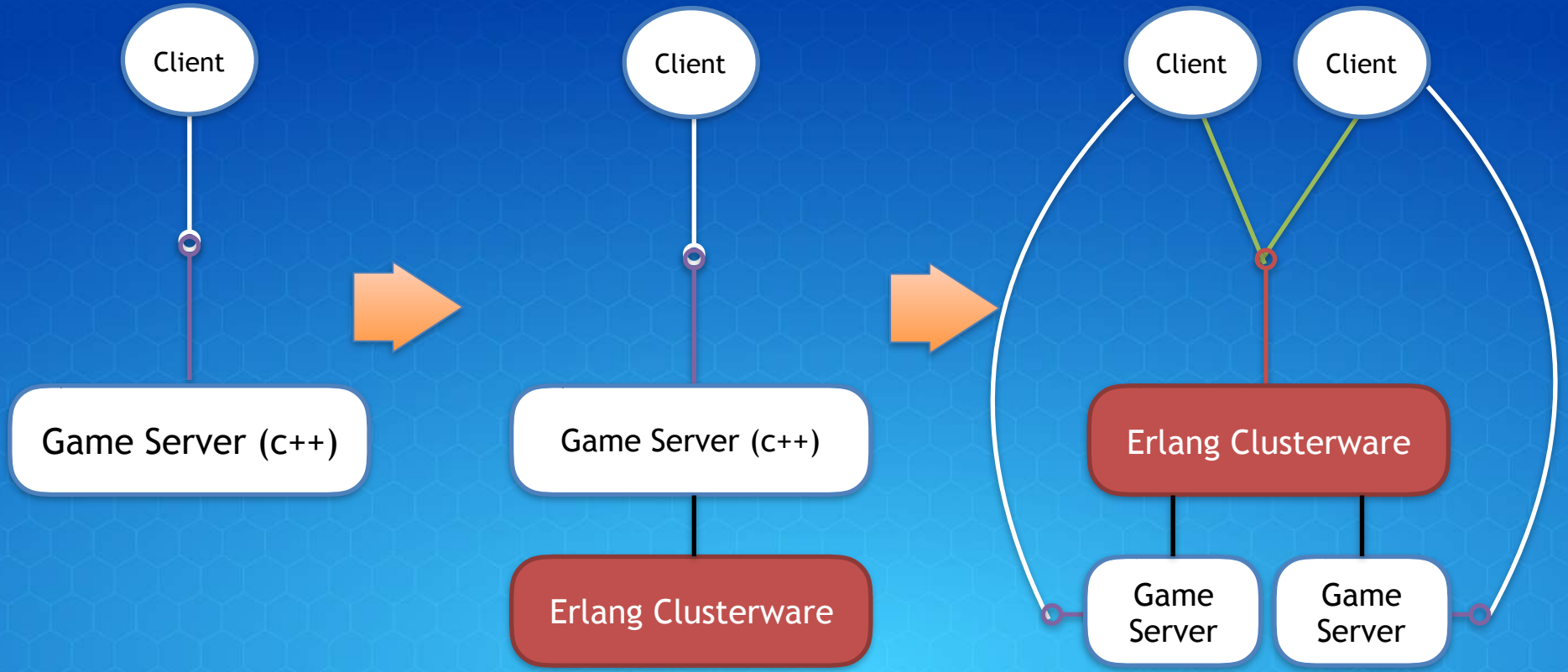
Base features (existing in Erlang)

- ✳ Game configuration (d
- ✳ Game ver ard)
- ✳ on
- ✳ management
- ✳ Game server upgrade
- ✳ Server scaling
- ✳ Load balance users per arena
- ✳ Use low cost infrastructure (AWS spot fleet)

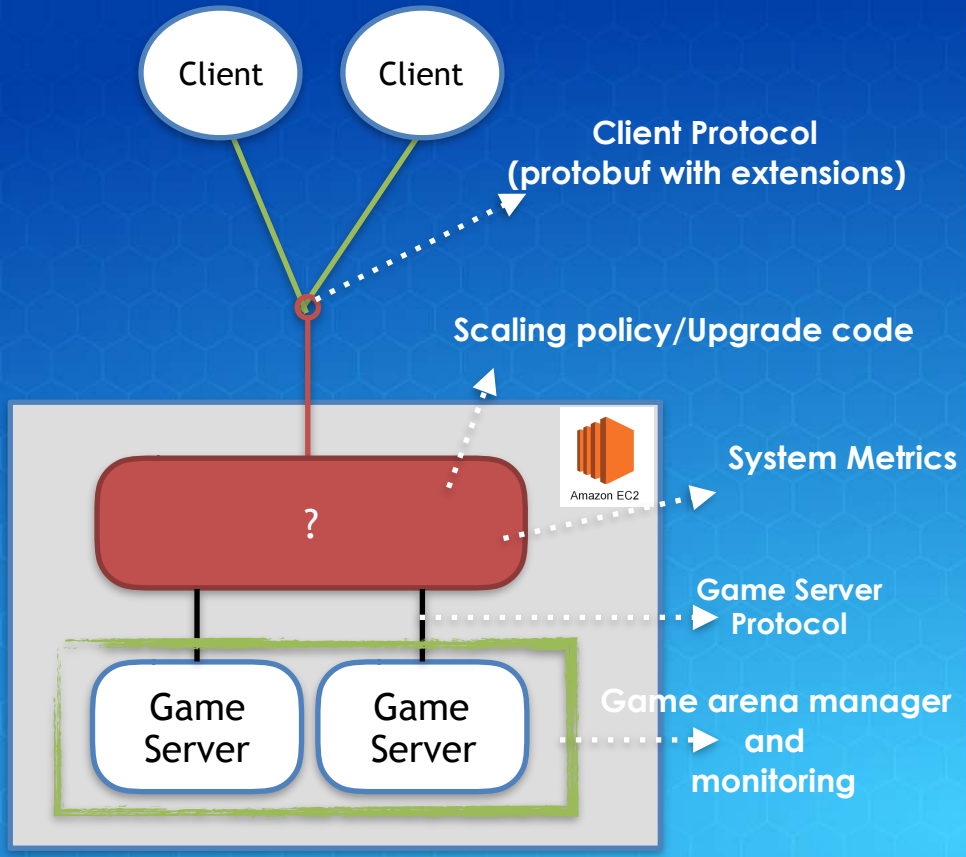
Time to market (2/3 months to release world wide)


HISTORY.COM

What is Maestro... (initial architecture)



What is Maestro... (the opportunity)

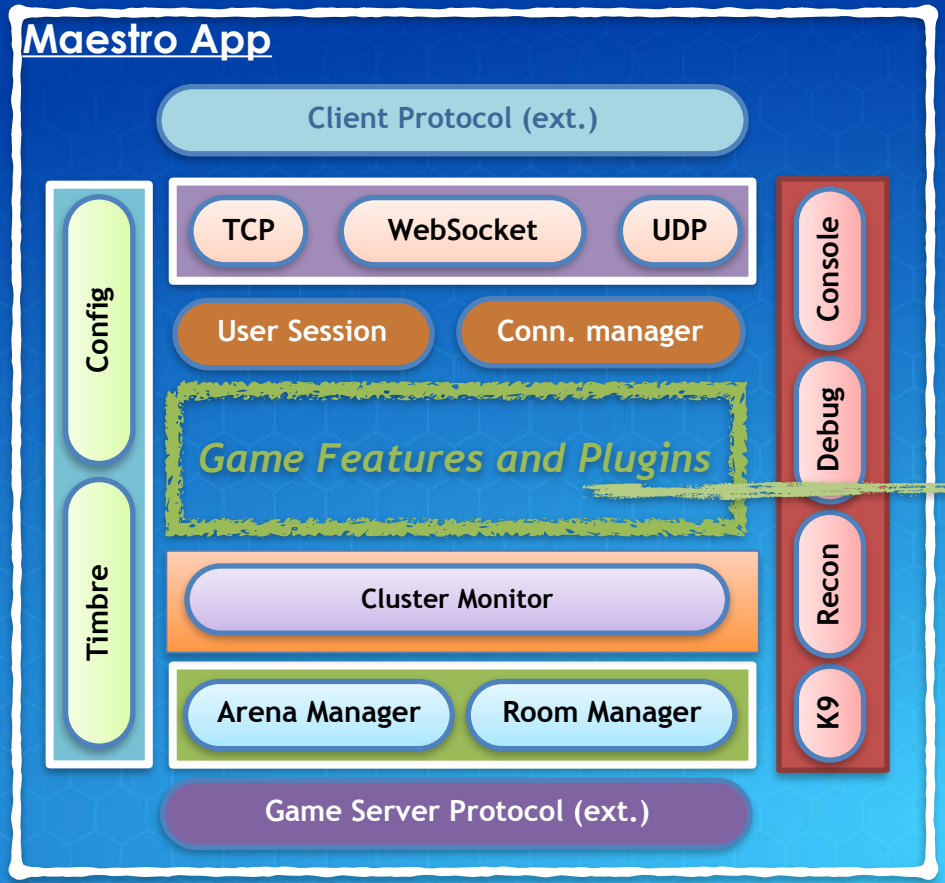


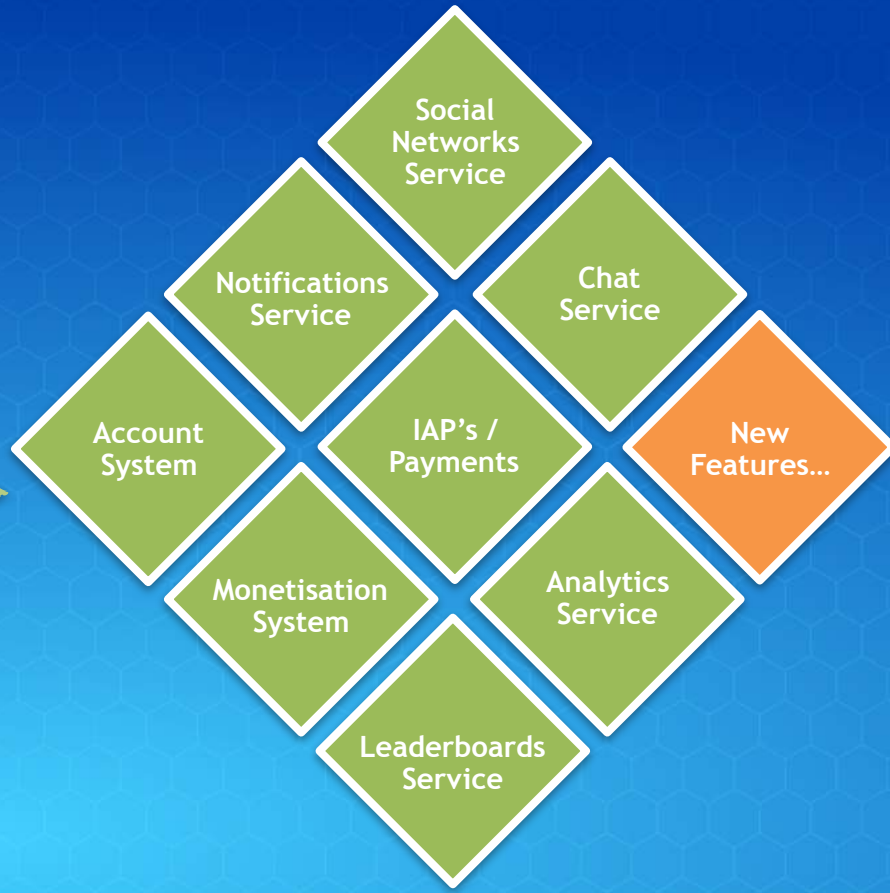
Our TODO shortlist

- Common protocol for client
- Common protocol for game server
- Handling client connection (TCP/UDP/WS)
- Report metrics to our monitoring system
- Allow to scale up and down the server infrastructure.
- Game server “code upgrade”
- Redirect players to game arenas
- This is a NEW APPROACH to IO Games, lets design to reuse “tomorrow”

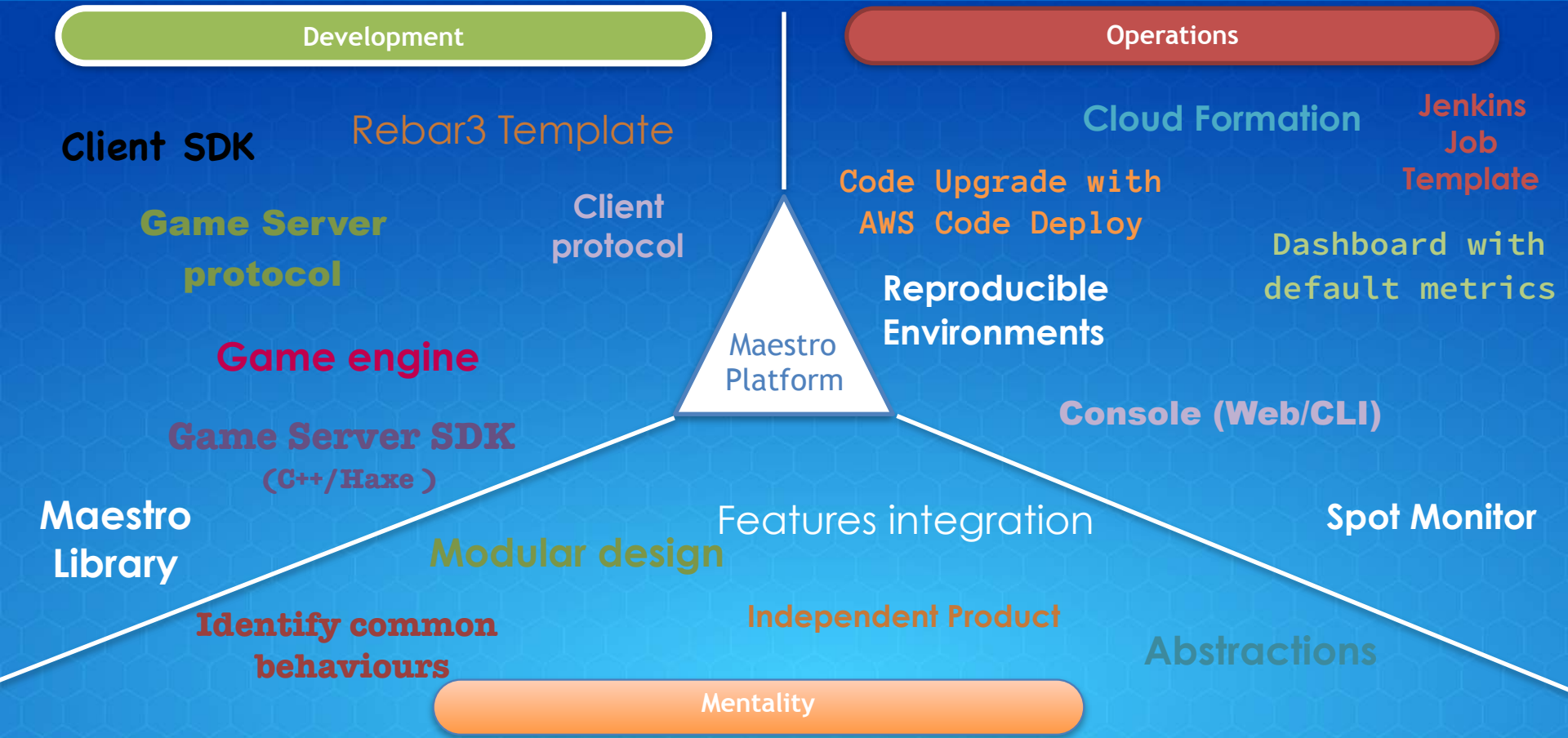

MAESTRO IS BORN...

Maestro as a Backend Product





Maestro Ecosystem



The Fundamentals

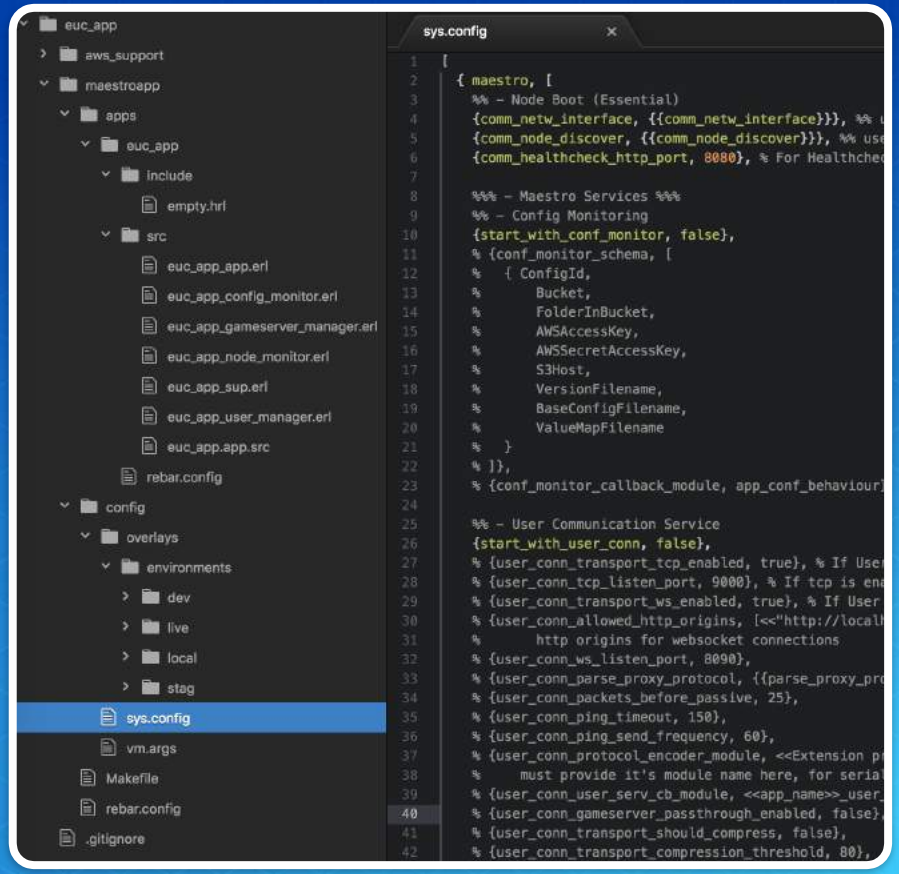
Maestro Library

- Manages core functionality
- Provides optional services
- Defines a communication protocol
- Defers decisions to callback modules
- Provides log and analytics facilities

Maestro App

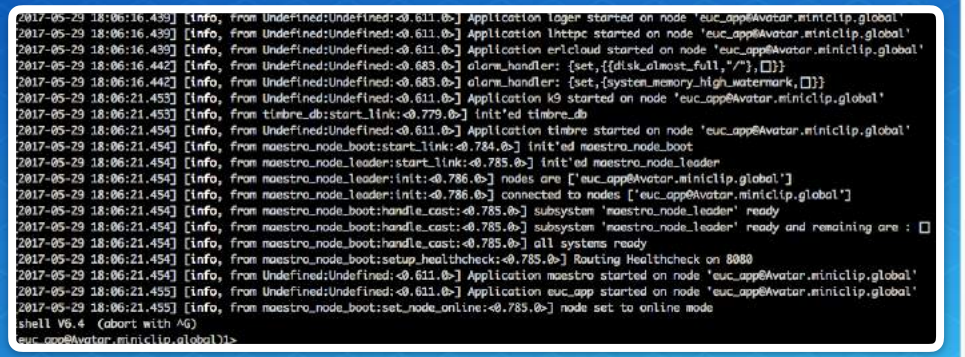
- Game specific
- Configures services to use
- Extends communication protocol
- Responds to callbacks
- Created through a rebar3 template

The Fundamentals



Starting a new Project

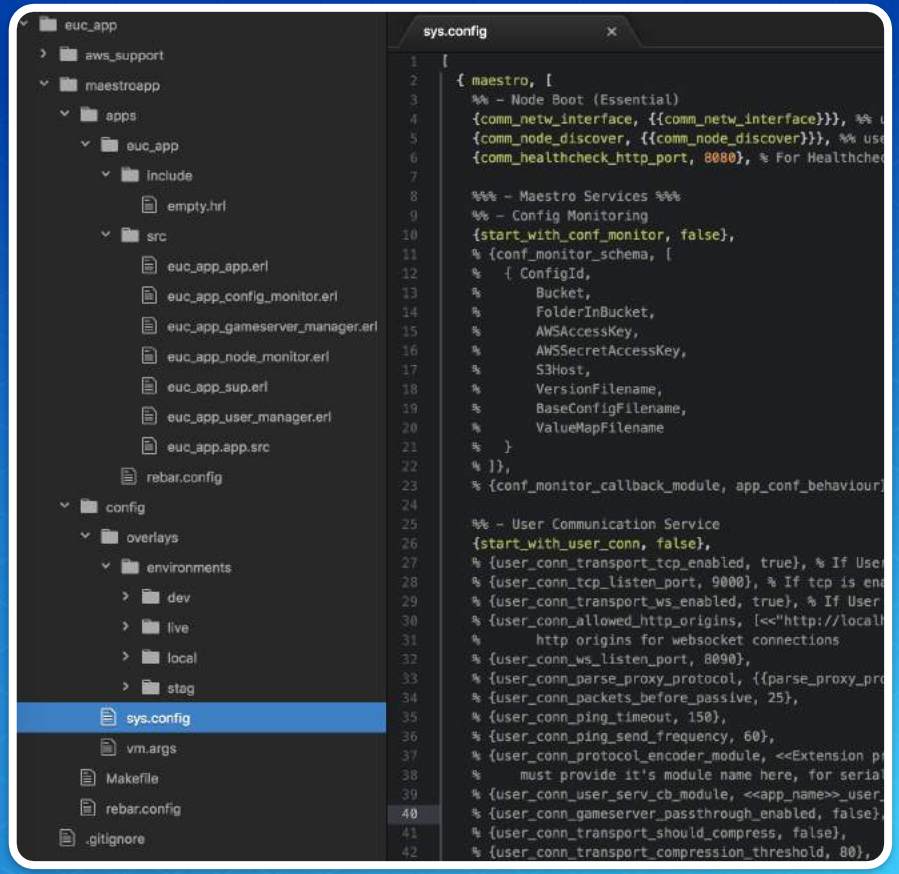
1. ‘rebar3 new maestroapp euc_app’
2. ???
3. Profit! (Maybe…)



Starting a new Project

All we have to do now is…

‘make run’



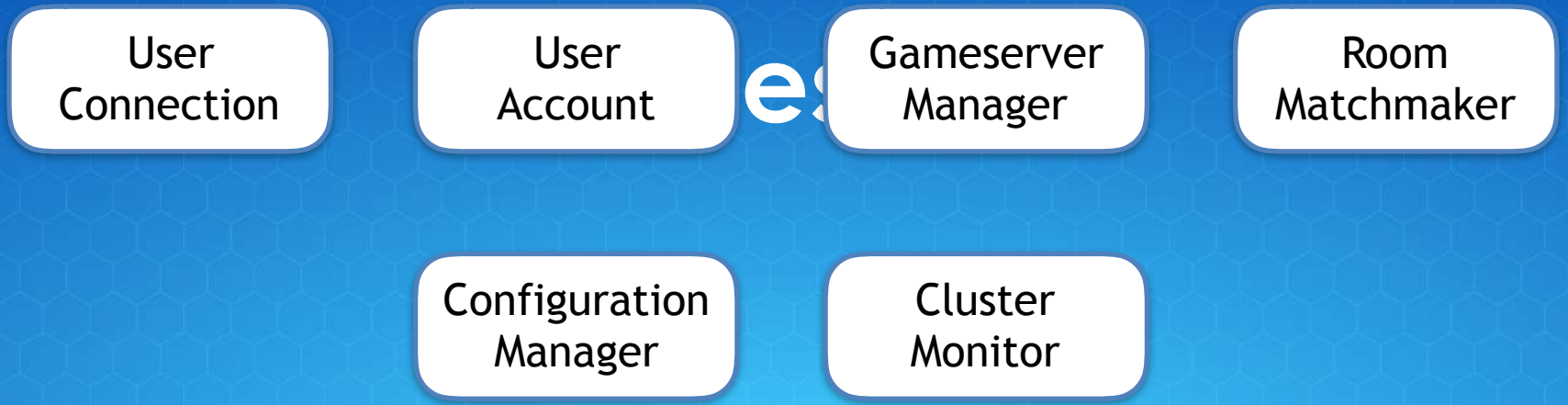
Configuring Services

By default, no services are enabled, only the bare minimum is running

Services are enabled and disabled through sys.config

The template provides the various available service options in comments

The Fundamentals



User Connection Service

Manages all communications between a maestro app and the client application

Can use TCP, UDP and/or Websocket as the underlying transport (using ranch and cowboy)

Implements the maestro user communication protocol

Delegates specific behaviour to the maestroapp through a callback module implementing the user connection behaviour

User Connection Service

Configuration:

```
%% --- User Communication Service
  %% Optional - default: false
  %% Must be true to initialise and use the service.
  {start_with_user_conn, false},

  %% Required
  %% Name of the module, in your specific maestroapp, that implements the behaviour 'maestro_users_serv_cb'.
  %% These callbacks will be called during normal User Conn Service execution. See documentation bellow for
  %% more info on these callbacks
  {user_conn_user_serv_cb_module, callback_module_name},

  %% Optional - default: false.
  %% If true, the service will accept clients through tcp.
  {user_conn_transport_tcp_enabled, false},

  %% Optional - default: 9000.
  %% Defines which port to listen to tcp clients on, if 'user_conn_transport_tcp_enabled' is true.
  {user_conn_tcp_listen_port, 9000},

  %% Optional - default: false.
  %% If true, the service will accept clients via websocket. (note that port used for websockets is defined in the
  %% mandatory config option, 'comm_http_port').
  {user_conn_transport_ws_enabled, false},

  %% Optional - default: false.
  %% Defines which port to listen to websocket clients on, if 'user_conn_transport_ws_enabled' is true.
  {user_conn_ws_listen_port, 8080},
```

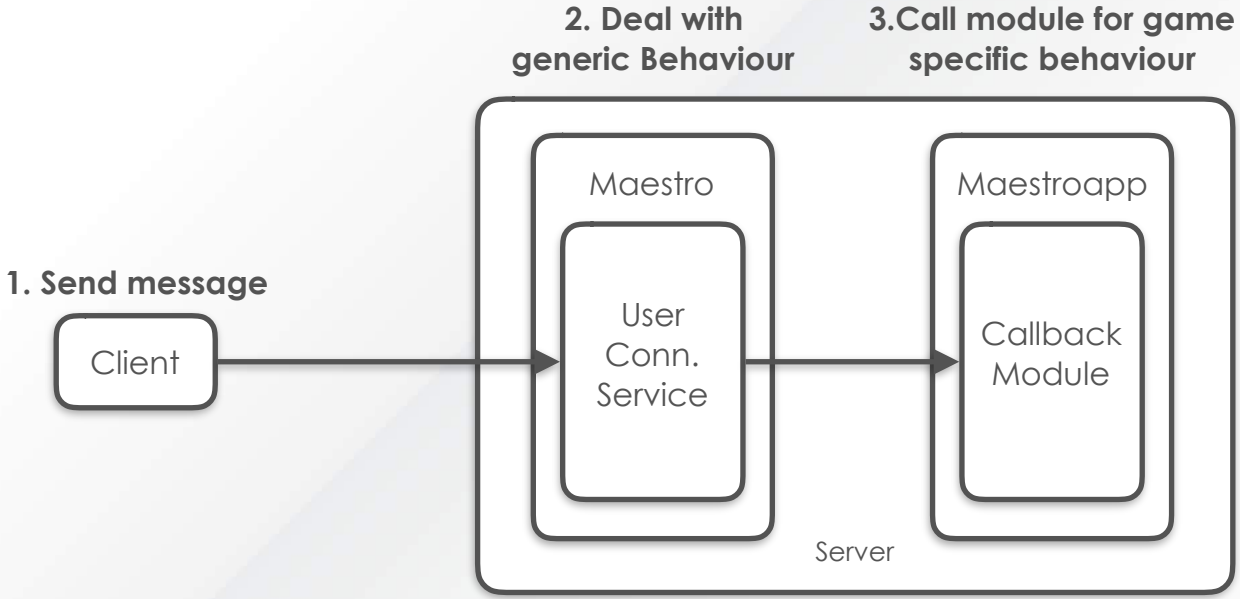
User Connection Service

Configuration:

```
%% Optional - default: 25
%% Used to define how many packets to read from socket before setting it to passive
{user_conn_packets_before_passive, 25},

%% Optional - default: 150
%% Amount of time to pass without receiving back a pong before disconnecting the user
{user_conn_ping_timeout, 150},

%% Optional - default: 50
%% Amount of time to wait between sending two ping messages
{user_conn_ping_send_frequency, 50},

%% Optional - default: false
%% If false, when the user wants to join a game, the User Conn Service will reply back to the user
%%  with an IP and Port to connect to in order to enter the game.
%% If true, when the user wants to join a game, the User Conn Service will create a socket to the
%%  gameserver, to pass on gameplay messages coming from the player.
{user_conn_gameserver_passthrough_enabled, false},

%% Optional - default: false
%% If true, Messages sent to the client will be compressed above a certain size limit. Compression is done by zlib.
{user_conn_transport_should_compress, false},

%% Optional - default: 80
%% When compression is enabled, this parameter customises the message size limit when compressing user messages
{user_conn_transport_compression_threshold, 80},
```

User Connection Service

Usual Behaviour:



User Connection Service

Callback Behaviour:

```
%% Required
%% Called whenever a user attempts to create a new session (right after opening the socket)
%% Should return true if the given Version and platform are acceptable for the current running maestroapp version, false otherwise.
%% Returning true will allow the session creation to continue, returning false will result in a disconnect of type 'not_authorized'
-callback is_version_allowed_on_platform(Version :: binary(), Platform :: maestro_user_pb:'user_platform_enum'()) -> boolean().

%% Required
%% Called whenever a user is allowed as per the 'is_version_allowed_on_platform' callback.
%% On a create session, if further treatment is required, this is the callback to do it. In case the create_session_request message
%%  is extended, you can patter match those extensions here and act accordingly.
%% Use the three available replies to either not add anything to the response to the client, add to the response message with your own
%%  extended response or disconnect the user.
%% UserStateAppData will initially be received as 'undefined', this can be filled with any data you want to use for this user and passed
%%  back into the user conn service. Any further callbacks pertaining to this user will carry it's own UserStateAppData.
%%  Think of this as the state in a gen_server.
-callback handle_create_session(ConnectionInfo :: maestro_users_sockserv:user_connection_info(),
Message :: maestro_user_pb:create_session_request()) ->
  {ok, UpdatedUserStateAppData :: term()} |
  {reply, CreateSessionResponseMsgExtension :: #{}, UpdatedUserStateAppData :: term()} |
  {reply_with_disconnect, Message :: maestro_user_pb:'disconnect.reason_enum'(), DisconnectMsgExtension :: #{}, UpdatedUserStateAppData :: term()}.

%% Required
%% Called whenever a user sends a message not in the base maestro protocol (a protocol extension message).
%% You can use this callback to treat any message you have extended the protocol with, using the data in your own UserStateAppData.
%% Similar to other user conn callbacks, you can have no reply, reply with any message or disconnect the user
-callback handle_user_message(Message :: #{}, UserStateAppData :: term()) ->
  {noreply, UpdatedUserStateAppData :: term()} |
  {reply, Message :: #{}, UpdatedUserStateAppData :: term()} |
  {reply_with_disconnect, Message :: maestro_user_pb:'disconnect.reason_enum'(), DisconnectMsgExtension :: #{}, UpdatedUserStateAppData :: term()}.


%% Optional
%% Called whenever the user socket disconnects. Use this to free any resources associated with the User, or to save any final permanent state
-callback handle_user_session_end(UserStateAppData :: term()) -> ok.
```

User Connection Service

After configuring the service, and implementing the callbacks, we're ready for liftoff. We now have a client acceptor and we can easily handle its behaviour.

Maestro will deal with the user connection boilerplate (according to Miniclip's standards), and we can fully focus on the domain logic of the project we're working on.

Gameserver Manager Service

The external game server manager is prepared to launch and monitor any external executable.

It implements the maestro control protocol in order to talk to its managed executables.

Delegates specific behaviour to the maestroapp through a callback module implementing the gameserver manager behaviour.

Gameserver Manager Service

Configuration:

```
%% External Gameserver Service
%% Optional - default: false
%% Must be true to initialise and use the service.
{start_with_external_gameserver, true},

%% Required
%% Name of the module, in your specific maestroapp, that implements the behaviour 'maestro_gameserver_manager_cb'.
%% These callbacks will be called when downloading new configuration files. See documentation below for more info on thes
{gameserver_callback_module, callback_module_name},

%% Required (only if not using S3 Gameserver Downloader)
%% Folder where the gameserver binary (and any other support files for it) will be placed for the service to check.
%% This folder will serve as a template for all launched gameservers. The service will copy this folder every time it
%% launches a new gameserver.
{folder_with_gameserver_binary, "path/to/gameserver"},

%% Optional - default: true
%% If true, the service will automatically monitor template for a new version.
{gameserver_auto_check_new_version, true},

%% Required
%% Amount of parallel gameservers to launch
{gameserver_instance_amount, 1},

%% Required
%% Port to connect to gameservers for control.
{gameserver_control_conn_listen_port, 8999},
```

Gameserver Manager Service

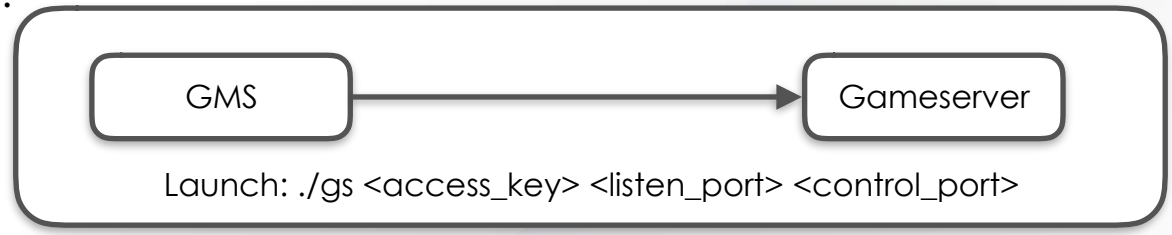
Configuration:

```
{folder_with_gameserver_binary, "path/to/gameserver"},

%% Optional - default: true
%% If true, the service will automatically monitor template for a new version.
{gameserver_auto_check_new_version, true},

%% Required
%% Amount of parallel gameservers to launch
{gameserver_instance_amount, 1},

%% Required
%% Port to connect to gameservers for control.
{gameserver_control_conn_listen_port, 8999},

%% Required
%% Initial range for gameserver listen ports. These ports will be passed on to the gameservers for them to open their cli
%% listeners on. A port given to gameserver 'n' will be gameserver_tcp_listen_port_range_start+(n-1).
{gameserver_tcp_listen_port_range_start, 9001},

%% Optional - default: 900000
%% Amount of time in miliseconds that is given to a gameserver to shut itself down after receiving a terminate message.
%% If it's not shutdown in this timeframe, it will be forcefully killed.
{gameserver_time_to_terminate, 900000},

%% Optional - default: 5000
%% Amount of time in miliseconds that is given to a gameserver to initiate a connection to the maestro control port.
%% If the gameserver doesn't respond in time, it will be shutdown.
{gameserver_detached_time_limit, 5000},
```

Gameserver Manager Service

Launching gameservers:

1.



2.

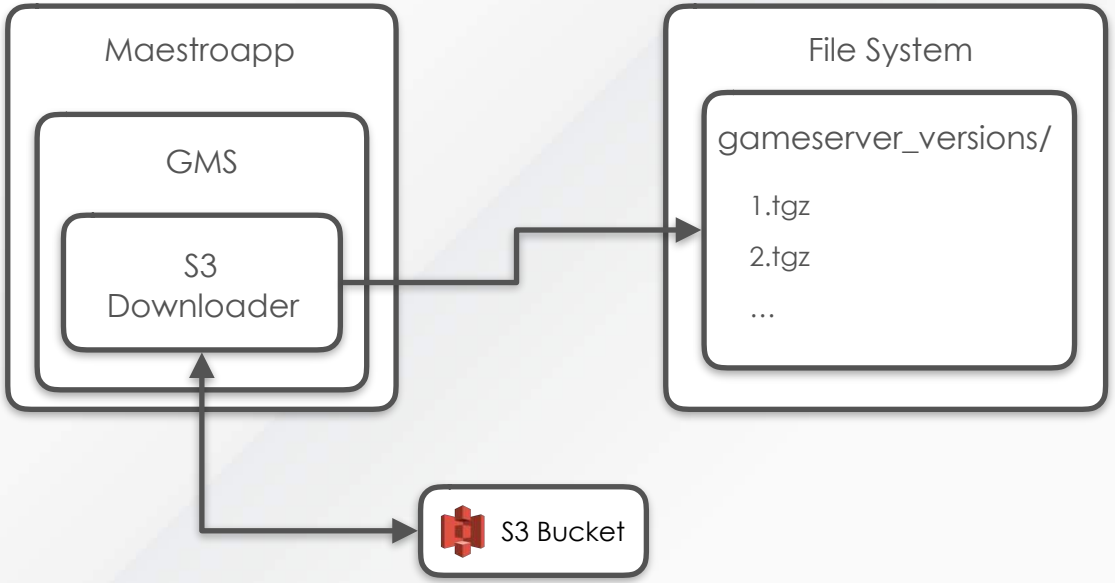
Gameserver Manager Service

Callback Behaviour:

```
%% gameserver on startup, in whatever way is decided for the project in question
%% (for example as arguments for the gameserver binary).
-callback provide_run_cmd(ServerName :: string(),
  TCPPortForMaestroComm :: non_neg_integer(),
  TCPPortForClientAccept :: non_neg_integer()) -> string().

%% Required
%% Should return a string with the bash command that will return the current version of the
%% gameserver binary.
-callback provide_version_check_cmd() -> string().

%% Required
%% Whenever the gameserver sends the regular state updates, this function will be called.
%% The state update message can be extended,
%% and this callback exists to deal with the extended data.
-callback handle_gameserver_state_update(GameserverKey :: string(), Message :: #{},
  GameserverStateExtendedData :: term()) -> term().

%% Required
%% In case the gameserver control protocol is extended, this is the callback that is called
%% whenever one of those extension messages arrives from the gameserver.
-callback handle_gameserver_message(GameserverKey :: string(), Message :: #{}) -> ok.
```

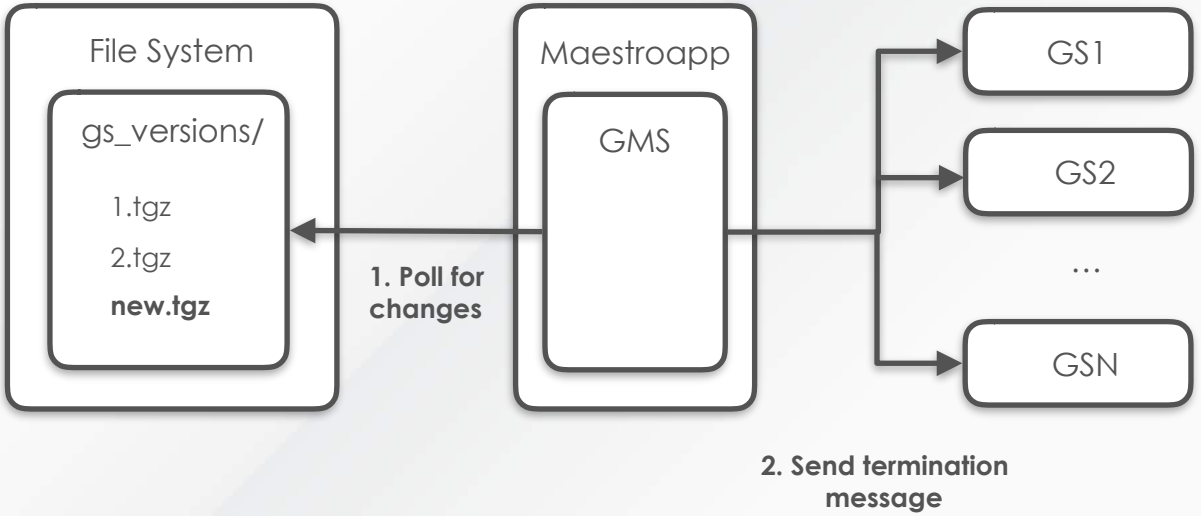
Gameserver Manager Service

The S3 Downloader sub-service:



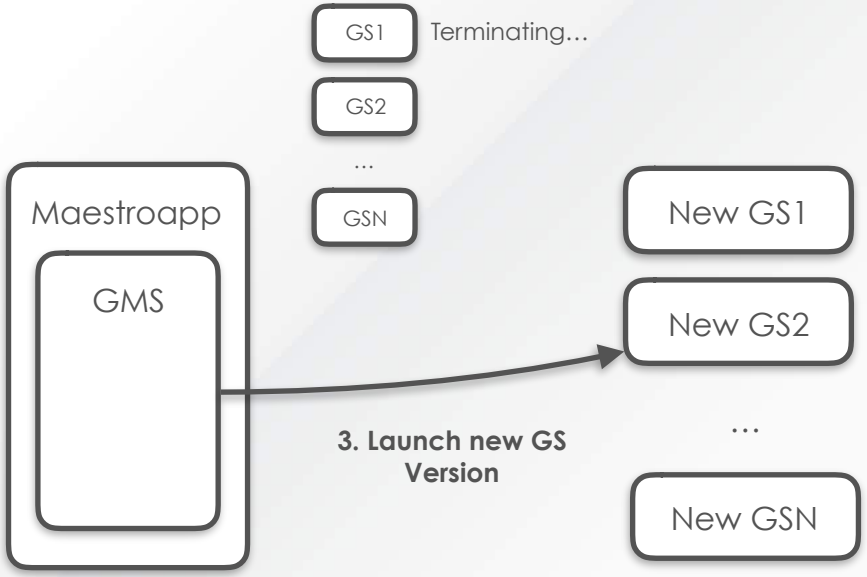
Gameserver Manager Service

Gameserver upgrade flow:



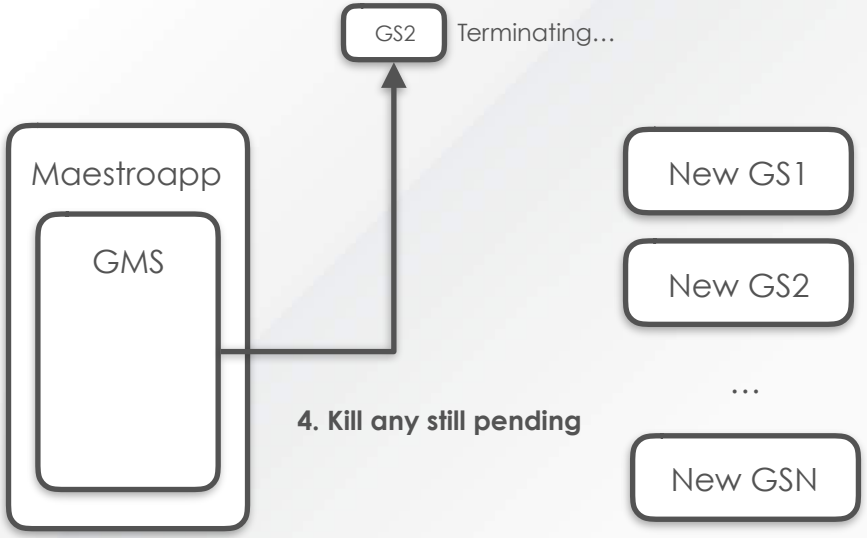
Gameserver Manager Service

Gameserver upgrade flow:



Gameserver Manager Service

Gameserver upgrade flow:



Gameserver Manager Service

Having the game server manager and S3 downloader up and running enables us to easily manage a stack of running game servers.

We can manage their behaviour, monitor their status, and deploy new revisions cleanly and automatically.

By following the maestro control proto and launch flow, any gameserver can be adapted to work with maestro.

Cluster Monitor Service

The Cluster Monitor Service is responsible for managing the erlang cluster of maestro apps running in several machines across an AWS based infrastructure.

It offers an API to fetch information about all nodes in the cluster, as well as an API for interacting with AWS (using erlcloud).

It deals with the automatic scaling of the infrastructure, basing it's decisions on responses from callbacks made on the maestro apps across the cluster.

Cluster Monitor Service

Configuration:

```
{start_with_node_monitoring, true},

%% Optional - default: undefined
%% Name of the module, in your specific maestroapp, that implements the behaviour 'maestro_node_monitor_cb'.
%% These callbacks can be registered to control scaling decisions for the cluster. See documentation bellow
%% for more info on these callbacks.
{user_monitor_cb_module, callback_module_name},

%% Required, if 'user_monitor_cb_module' is undefined
%% If using defaults (not defining a cb module), this is the value used to gauge how many players should be
%% playing per gameserver at max capacity.
{game_max_users_per_game, 100},

%% Required, if 'user_monitor_cb_module' is undefined
%% If using defaults (not defining a cb module), this is the value used to gauge how many gameservers should
%% be running at max capacity.
{game_max_games_per_server, 3},

%% Required, if 'user_monitor_cb_module' is undefined
%% If using defaults (not defining a cb module), this is the value used as a margin of error for the amount
%% of playing users.
{game_autoscale_users_spread, 10},

%% Required
%% Amount of time between node monitor load checks
{node_load_check_interval, 600000},
```

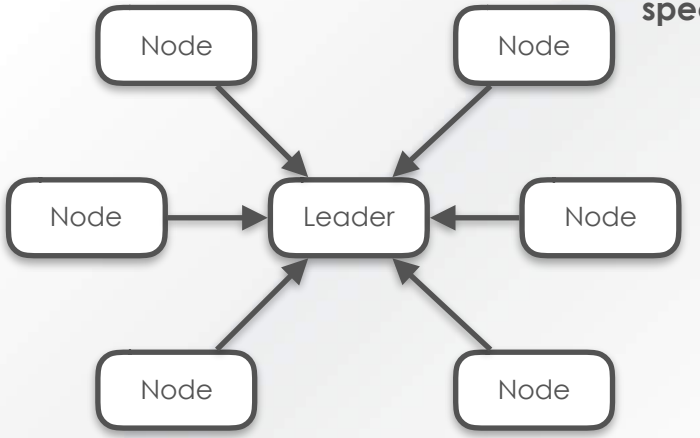
Cluster Monitor Service

Configuration:

```
%% Required, if 'user_monitor_cb_module' is undefined
%% If using defaults (not defining a cb module), this is the value used to gauge how many players should be
%% playing per gameserver at max capacity.
{game_max_users_per_game, 100},

%% Required, if 'user_monitor_cb_module' is undefined
%% If using defaults (not defining a cb module), this is the value used to gauge how many gameservers should
%% be running at max capacity.
{game_max_games_per_server, 3},

%% Required, if 'user_monitor_cb_module' is undefined
%% If using defaults (not defining a cb module), this is the value used as a margin of error for the amount
%% of playing users.
{game_autoscale_users_spread, 10},

%% Required
%% Amount of time between node monitor load checks
{node_load_check_interval, 600000},

%% Required
%% Amount of time allotted to terminate a node
{node_termination_check_interval, 5000},

%% Required
%% Time interval for nodes to send their status to the leader
{node_stats_send_interval, 5000}
```

Cluster Monitor Service

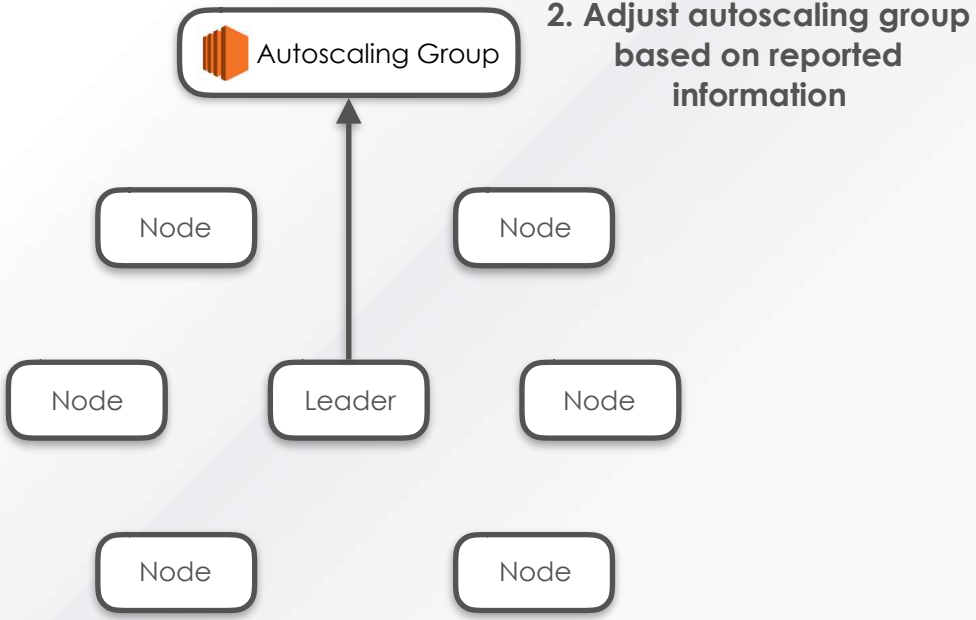
Scaling process:



1. Report game specific data to Node Leader

Cluster Monitor Service

Scaling process:



Cluster Monitor Service

Callback Behaviour:

```
%% Required
%% Callback triggered on each node in the cluster, allows the maestro app to
%% report any data it wants to the leader.
%% This data can be accessed via the leader callback, and can be used,
%% alongside the default information on number of gameservers and player to
%% perform scaling decisions.
-callback report_node_data_to_leader() ->
  CustomNodeData :: term().

%% Required
%% Callback triggered on only the node leader of a cluster.
%% Using a list of information from each node, alongside it's custom data
%% reported via 'report_node_data_to_leader', the leader must reply no_change,
%% if current load is fine, add_nodes, indicating an amount, if new instances
%% need to be spooled up, or kill_nodes with the indication of nodes to kill,
%% when running at over capacity.
-callback leader_scaling_check([{Node :: atom(), CustomNodeData :: term()}]) ->
  {no_change, ok} |
  {add_nodes, NrOfNodes :: non_neg_integer()} |
  {kill_nodes, NodesToKill :: [Node :: atom()]}.
```

Cluster Monitor Service

Node Monitor also has hooks to interact with other services, if they are running.

If a node is set to be killed, its Cluster Monitor Service will set itself to ‘offline’, which will:

- Inform the User Connection Service to stop accepting connections.
- Inform the Gameserver Manager Service to terminate its gameservers so that they drain connections.

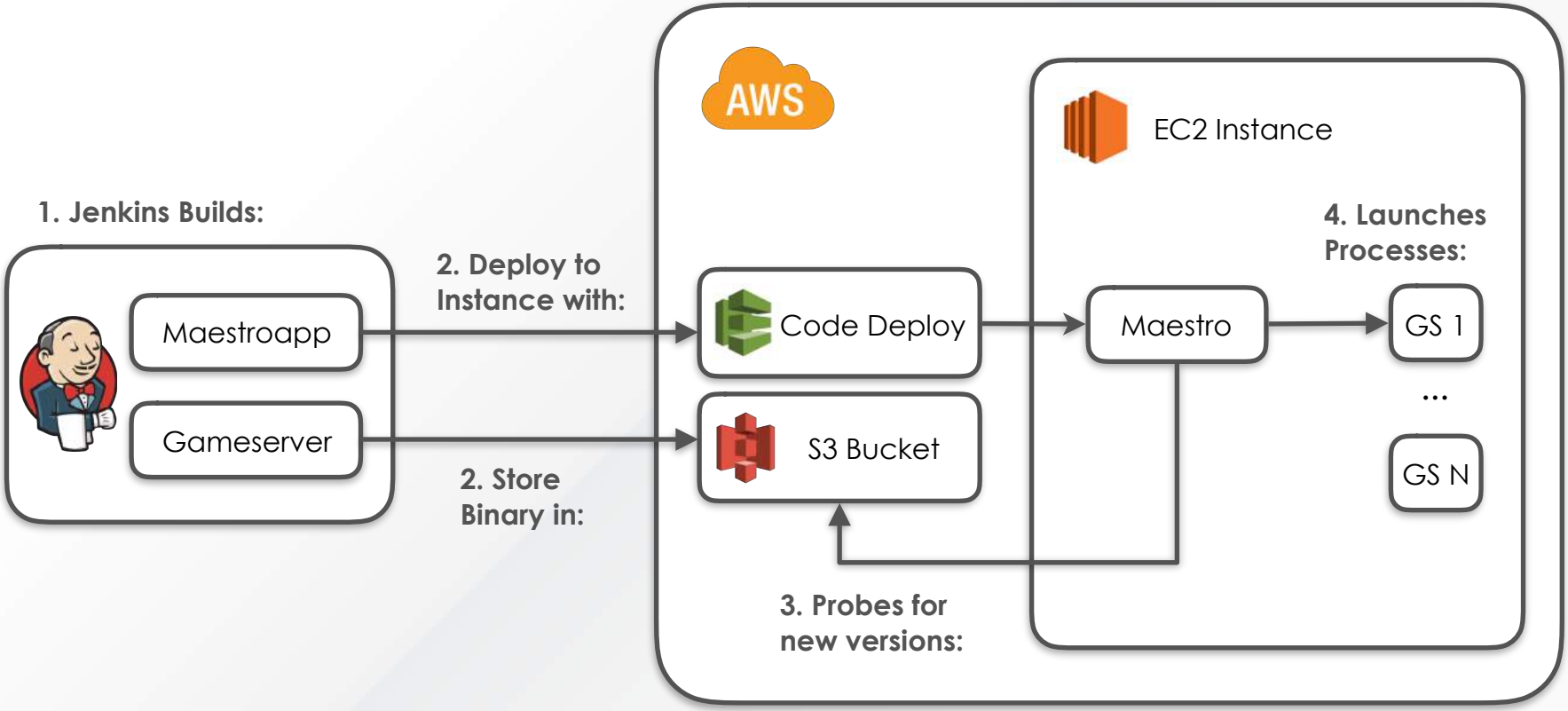
Cluster Monitor Service

After setting up the cluster monitor service, and it's scaling rules, we're ready to deploy an infrastructure of several interconnected maestroapps, each with their own gameservers.

The infrastructure is ready to grow, or shrink, according to user demand according to project specific rules that can be tailored to its needs.

Deployment and Monitoring

Deployment


AWS
EC2 Instance
1. Jenkins Builds:
Maestroapp
Gameserver
2. Deploy to Instance with:
2. Store Binary in:
Code Deploy
S3 Bucket
3. Probes for new versions:
Maestro
4. Launches Processes:
GS 1
...
GS N

Monitoring

AWS

EC2 Instance

GS 1 → Maestro

...

GS N

1. Aggregate Metrics

DATADOG

Realtime Dashboard & Monitors

2. Report to Web Dashboard

pagerduty

Incident reporting tool

3. Trigger Alarms

Maestro Ecosystem

Maestro Ecosystem

Partiture

Gameserver Maestro SDK

Provides compliance with maestro startup flow and communication.

Virtuoso

Client Maestro SDK

Common maestro features with hooks for game specific information.

Orchestra

C++ Gameserver Engine

Libuv based, maestro compliant gameserver engine.

Stagecraft

C++ Game Logic Framework

Component based game development toolchain.

Trade-offs and drawbacks

- As with every centralised code-base, a much greater deal of coordination must now happen between backend developers.
- When a completely new feature needs to be added to a project there is always going to be the question of 'should it be part of Maestro?'
- Maestro has its own workflows, that differ from our past products (callback based development, new type of protocol, etc…). Bringing new developers into maestro projects requires some adjustments.
- If (or when) any **major** re-work of a core system needs to be done to maestro, can have a ripple effect on the entire ecosystem, forcing several libraries to apply the changes

Final thoughts…

- Maestro platform enable us to develop a game with base features out-of-the-box just by pressing some “buttons”.
- We had ONE backend engineer to develop and operate 5 games.
- Is a re-usable and extensible backend product.
- Core features are provided by Maestro. We can integrate any game server developed internally and/or externally using current or new developed server SDK’s.
- Maestro intends to be our internal ‘Game as a Service’ platform.
- It provides the infrastructure and the toolchain to: develop, build, deploy and operate our games backend.

Q & A

Thank you

… we are hiring! http://corporate.miniclip.com