digium
The Asterisk Company

Scalable Asterisk Servers in a Large SIP Infrastructure

Matt Jordan
@mattcjordan

Or:

Thoughts on building a SIP network with Open Source tools as told from the perspective of an Asterisk guy who likes to employ JavaScript / Ruby / Python / Java / Go / Rust (non-C) developers

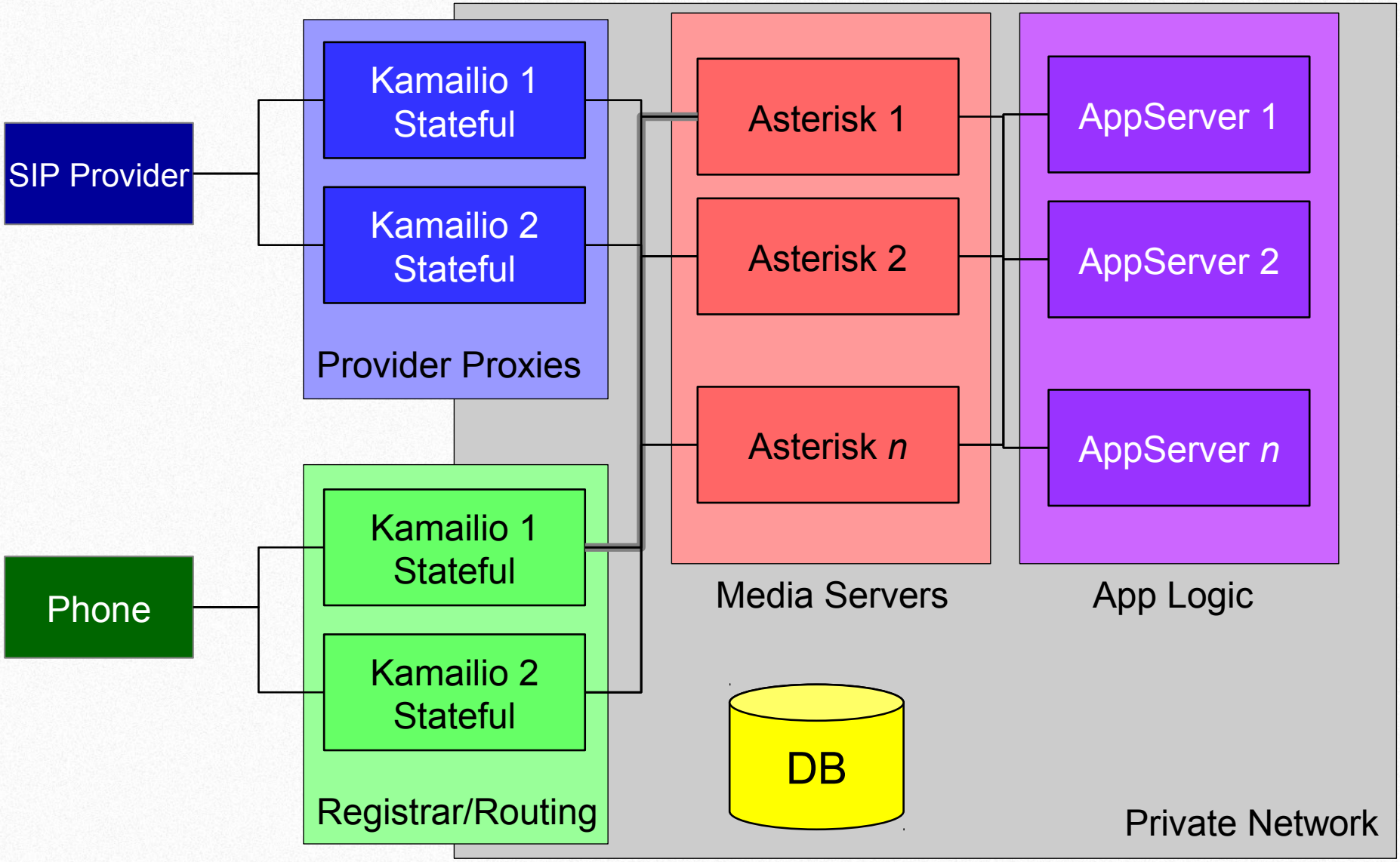
Goals

- Be able to scale out, then up
- Reasonable redundancy everywhere
- All applications are cattle, not pets
- Minimize the necessity of specialist monolith knowledge

Components

	Kamailio	Asterisk	Service
Scalability	**Great**	**Meh**	**Good***
API	**Meh**	**Good**	**Good***
Required Knowledge	**Bad**	**Bad**	**Good**

General System Architecture



Four General Problems

- Inbound SIP Registrations
  - How do phones find each other?

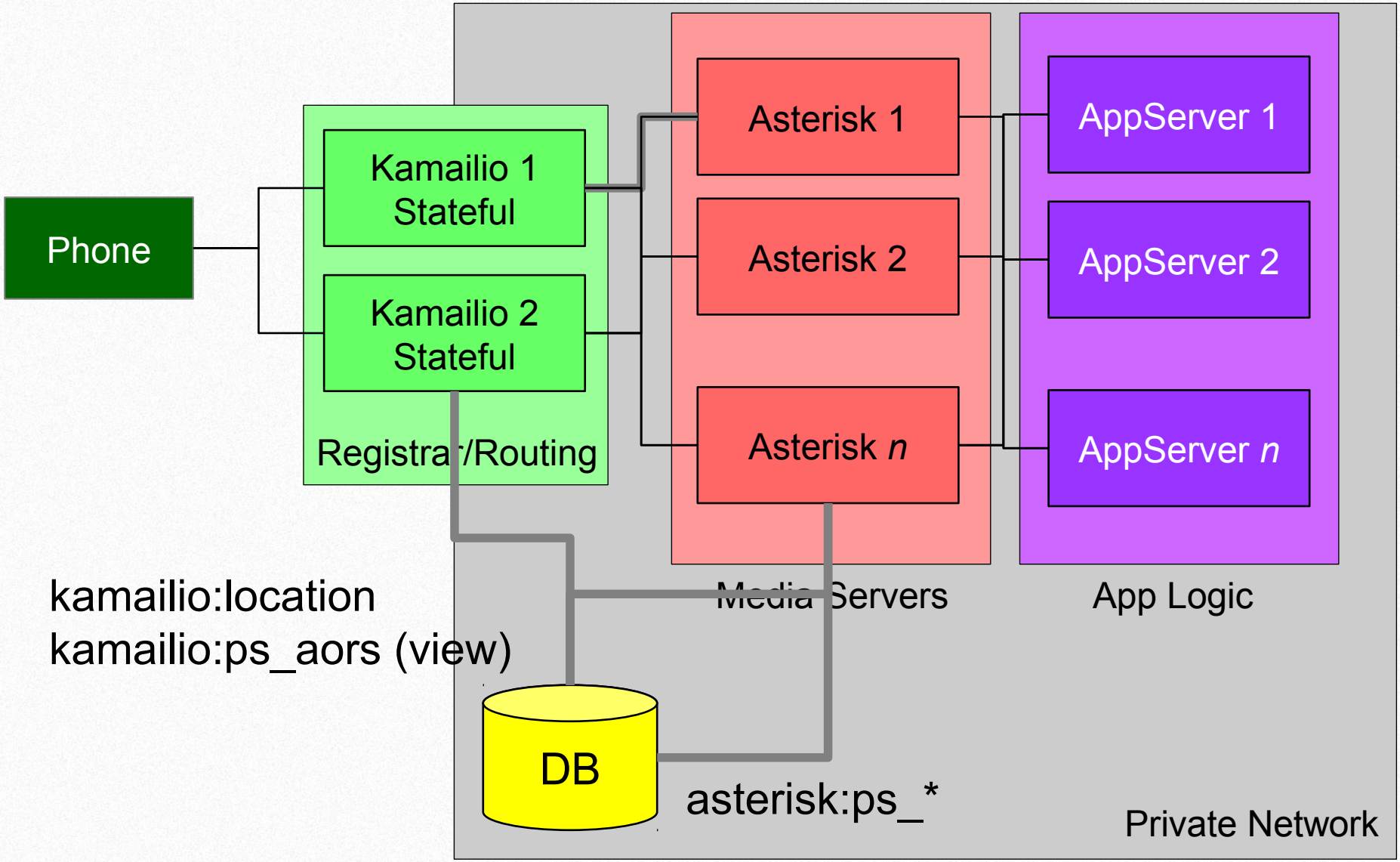
Handling REGISTER requests

- Goal 1: Use Kamailio
  - It scales better
  - Asterisk does not cluster registration state
- Goal 2: We don't want to:
  - Associate a SIP registration (phone location) to an Asterisk instance
  - Send all SIP registrations to all Asterisk instances

Handling REGISTER requests

- Kamailio is easy
  - Save the location using the registrar module
  - Fork a received REGISTER request to the other Kamailio instances so they can update their in-memory information
- Asterisk: Two Approaches
  - Use a view on the kamailio DB and look up location information by AoR (dial by AoR)
  - Use a “sidecar service” to expose location information (dial by URI)

Handling REGISTER requests: View



Handling REGISTER requests: View

- For each Asterisk instance:
  - Configure an ODBC connection to the Kamailio database
  - Create an Asterisk ps_aors realtime/ODBC mapping to the Kamailio database

```
ps_aors => odbc,kamailio
```

  - Configure sorcery to map aors to the ps_aors object from the realtime connection

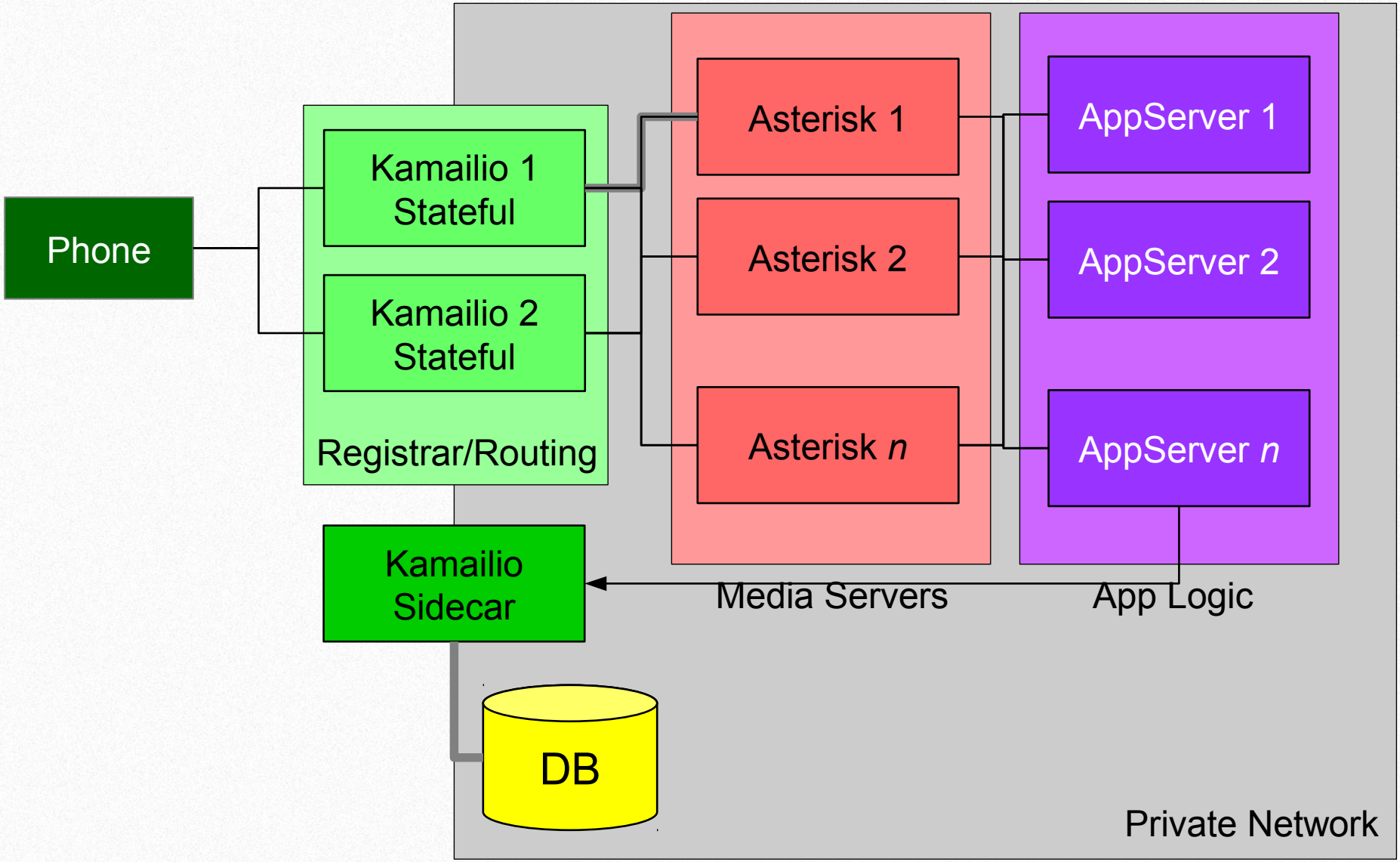
```
aor = realtime,ps_aors
```

- Pros: Can Dial by AoR from everywhere

```
; Assumes our AoR is named the same as the endpoint alice
same => n,Dial(PJSIP/alice)
```

- Cons: Creates a non-intuitive link between Asterisk and Kamailio

Handling REGISTER requests: Sidecar



Handling REGISTER requests: Sidecar

- Write a “sidecar” service to Kamailio
  - Expose a REST API that provides the location of a phone
  - Get the location of the phone from the AppServer
    - Error return codes can indicate the lack of a registered contact
  - Dial by URI:

```
const uri = getContactForEndpoint(alice);
const dialstring = `PJSIP/alice/${uri}`;
```

- Pros: Relationships are explicit in code; failures are explicit
- Cons: Have to write a service; not easily used from Asterisk dialplan

Four General Problems

- Inbound SIP Registrations
  - How do phones find each other?
- Routing/Distribution
  - How do we get the right SIP request to the right media server?

Routing/Distribution

- Option 1: Tie a customer domain to an Asterisk instance
  - Pros: Easy – all calls end up on the same media server
  - Cons: Doesn't scale, inefficient
- Option 2: Just round-robin using Kamailio dispatcher
  - Pros: Efficient, makes use of resources, tolerant to failure
  - Cons: Pushes the burden onto Asterisk and the application logic to get calls on the right servers

Four General Problems

- Inbound SIP Registrations
  - How do phones find each other?
- Routing/Distribution
  - How do we get the right SIP request to the right media server?
- Applications
  - How do we quickly iterate on application features to serve the never ending demands of our customers?

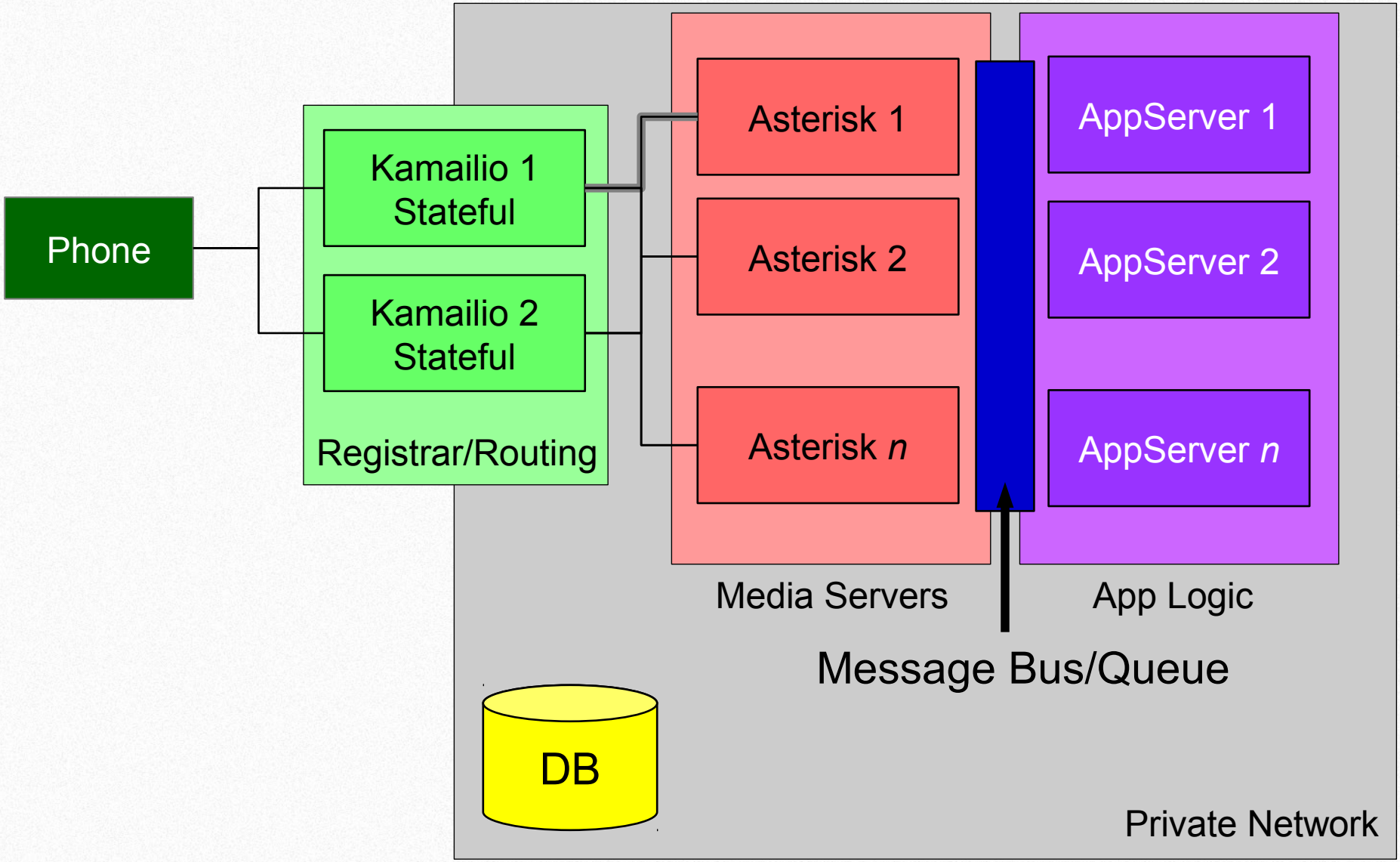
Applications: Things Get Tricky

- All calls will be arriving at random Asterisk instances
  - Customer information cannot 'live' or be 'owned' by Asterisk
    - Endpoint definition
    - Prompts
    - Recordings
  - May have to move a channel to another Asterisk instance for some application specific scenarios
- Keep the dialplan simple!
  - Cannot scale a customer specific dialplan
  - Cannot maintain a complex dialplan

Applications: Things Get Tricky

- Phase 1: Use AGI as a Dispatcher
  - Easiest to get started
  - Load balances with `hagi` protocol
  - Can selectively replace dialplan applications with ARI
  - Pros: Easy to get started, can choose when to replace things
  - Cons: Requires special DNS entries
- Phase 2: Use ARI with a proxy/middleware layer
  - General idea: terminate websockets locally and pass event messages over a message queue
  - Provide a facade that allows a rules engine to claim 'ownership' over some channel based on the application it entered
  - Pros: Fully decoupled
  - Cons: Lots of custom dev work

Handling REGISTER requests: Sidecar



Applications: Things Get Tricky

- Endpoints
  - Map the PJSIP objects to a realtime backend
  - Use a cache!

```
endpoint/cache = memory_cache,
                 maximum_object=1024,
                 object_lifetime_maximum=3600
endpoint = realtime,ps_endpoints
```

- Prompts
  - Create a REST service that hosts custom sound files for customers
  - Use remote URI playback to play the media
    - Will temporarily cache locally on each Asterisk instances

Applications: Things Get Tricky

- Recordings
  - Use ARI to manipulate recordings
  - `StoredRecording` object exposes a REST route to retrieve the media from the Asterisk instances
- Moving Channels
  - Sometimes a channel needs to get co-located on the same Asterisk instance as an already existing channel
    - Multi-party conferences
    - Call Queues
  - Use the `redirect` ARI command
    - New request URI must contain information that the application(s) can use logically (such as a token)

```
sip:{stateTokenId}@{correctAsteriskInstanceIP}
```

Four General Problems

- Inbound SIP Registrations
  - How do phones find each other?
- Routing/Distribution
  - How do we get the right SIP request to the right media server?
- Applications
  - How do we quickly iterate on application features to serve the never ending demands of our customers?
- Subscriptions
  - How do we make blinky lights flash?

Handling Subscriptions

- General Approach
  - Use Kamailio as the Subscription Server
    - For the same reason as it makes a good registrar
  - Use Asterisk to generate state notifications
    - PUBLISH to the Kamailio instances
    - Kamailio will NOTIFY subscribed User Agents
- Kamailio is easy
  - Handle SUBSCRIBE requests using presence module in a similar fashion to REGISTER requests
  - Fork received SUBSCRIBE requests to other instances in the pool to have them update in-memory information
  - Handle PUBLISH requests in the standard fashion

Handling Subscriptions

- Asterisk: PUBLISH to the Kamailio cluster
  - Use auto-hints to generate hints intelligently when a device state change occurs
  - Define an outbound publish to the Kamailio cluster
    - Use DNS SRV to round-robin across the cluster

```
[test-esc]
type=outbound-publish
server_uri=sip:kamailio-instances@mydomain.com
from_uri=sip:asterisk_ip
event=dialog
multi_user=yes
@body=application/dialog-info+xml
```

Questions

Useful Links

- PJSIP Realtime
  https://wiki.asterisk.org/wiki/display/AST/Setting+up+PJSIP+Realtime
- PJSIP Dialing
  https://wiki.asterisk.org/wiki/display/AST/Dialing+PJSIP+Channels
- Sorcery Caching
  https://wiki.asterisk.org/wiki/display/AST/Sorcery+Caching
- Configuring Asterisk for PUBLISH to Kamailio
  https://wiki.asterisk.org/wiki/display/AST/Configuring+res_pjsip+for+Presence+Subscriptions
- ARI Examples and Tutorials
  https://wiki.asterisk.org/wiki/pages/viewpage.action?pageId=29395573