Connecting people: WebRTC and the Pexip collaboration platform

TF-WebRTC : May 19, 2015

pexip

Who are Pexip?

- Founded in April 2012
- Strong video heritage
- Manufacture Infinity – a pure-software, virtualized, distributed, scalable collaboration platform

pexip

Who am I?

- John-Mark Bell
- Software engineer in Pexip's UK R&D team
- Been with Pexip since day one
- Responsible for Pexip's OS & product security
- Implemented Pexip's initial WebRTC stack

pexip

Collaboration technologies

pexip

Some important trends

- Hardware -> Software
- Personalisation & BYoD
- User driven demand

It's no longer about the technology

pexip

Where does WebRTC fit?

- Browser-based clients are (becoming!) universal
- Can be used in isolation, or to augment traditional voice/video solutions
  - Screen/presentation sharing
  - Control functionality
- May enhance existing web collaboration tools

Opens up novel solutions and use-cases

pexip

Pexip's philosophy

- Interoperability matters
  - People want to use whatever client/device they have
  - Collaboration with people using different technologies needs to work (within and between organisations)
- Scale is inevitable
  - Ease of access to capable clients drives demand
  - Generational change – people expect to use this stuff

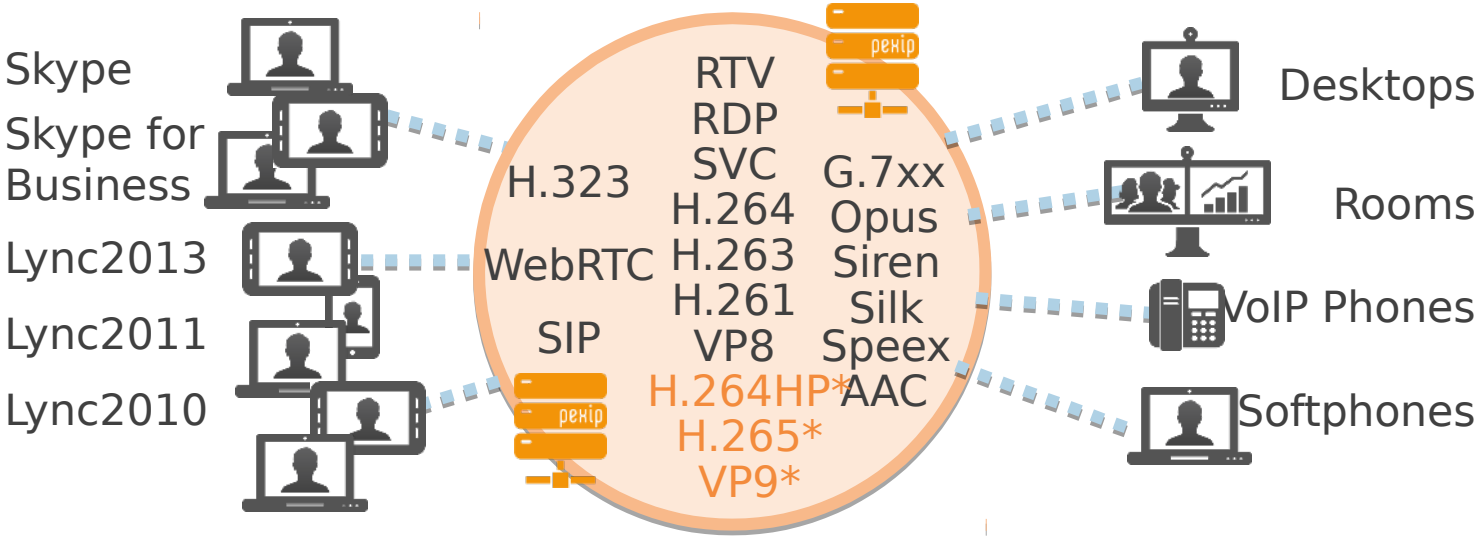
pexip

Pexip's philosophy

- Extensibility is vital
  - We provide a solid, feature-rich foundation
  - But we don't do everything or solve every use case
  - Therefore, we expose open, documented APIs which allow an ecosystem to build around us
  - This also allows customers to be proactive in addressing their specific needs by building on our foundations

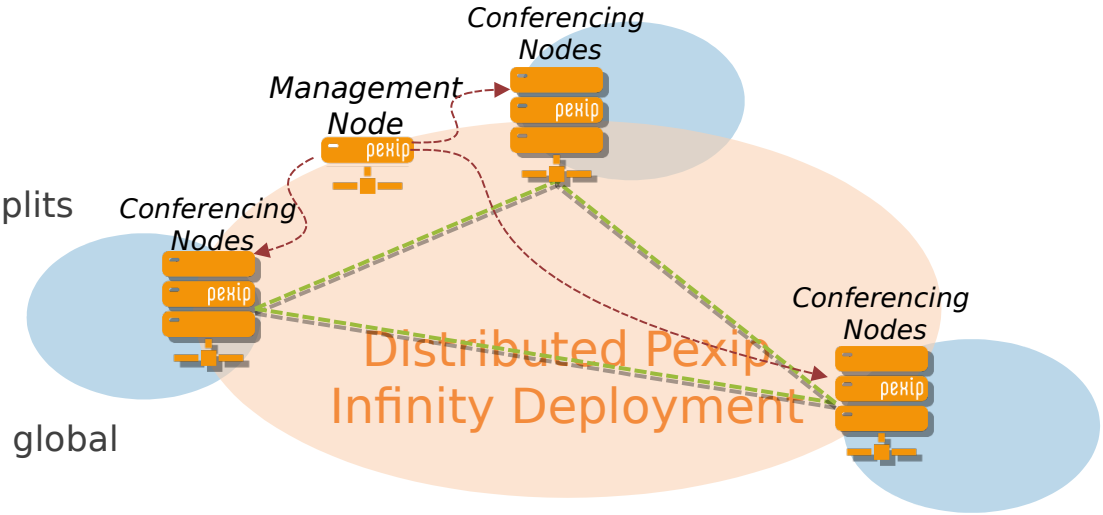
pexip

Interoperability



Pexip Distributed Architecture

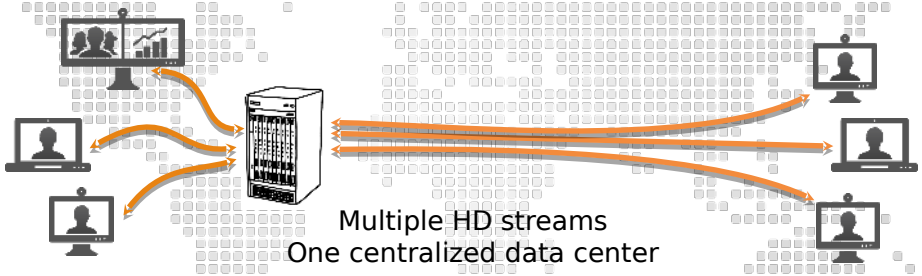
- Typical Deployment
  - 1 x Management Node
  - Multiple Conferencing Nodes
- Resilience
  - Resilient to temporary network splits
  - Leverages VMware and Hyper-V resilience mechanisms
- Benefits
  - Management: Single point of management & diagnostics for a global deployment
  - Keeps bandwidth usage local
  - Minimizes bandwidth usage between regions



pexip

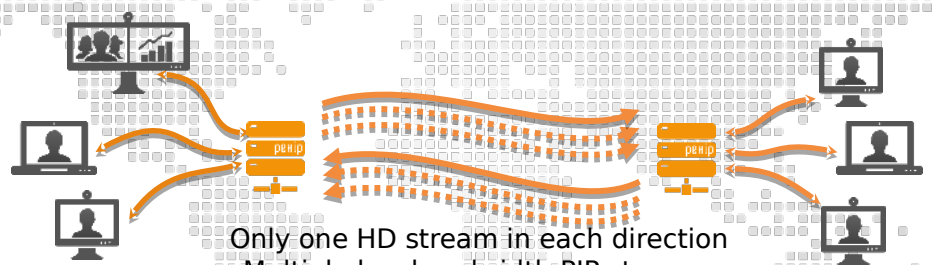
Distributed Architecture – Conferencing

Traditional Centralized Deployment



Multiple HD streams
One centralized data center

Distributed Conferencing

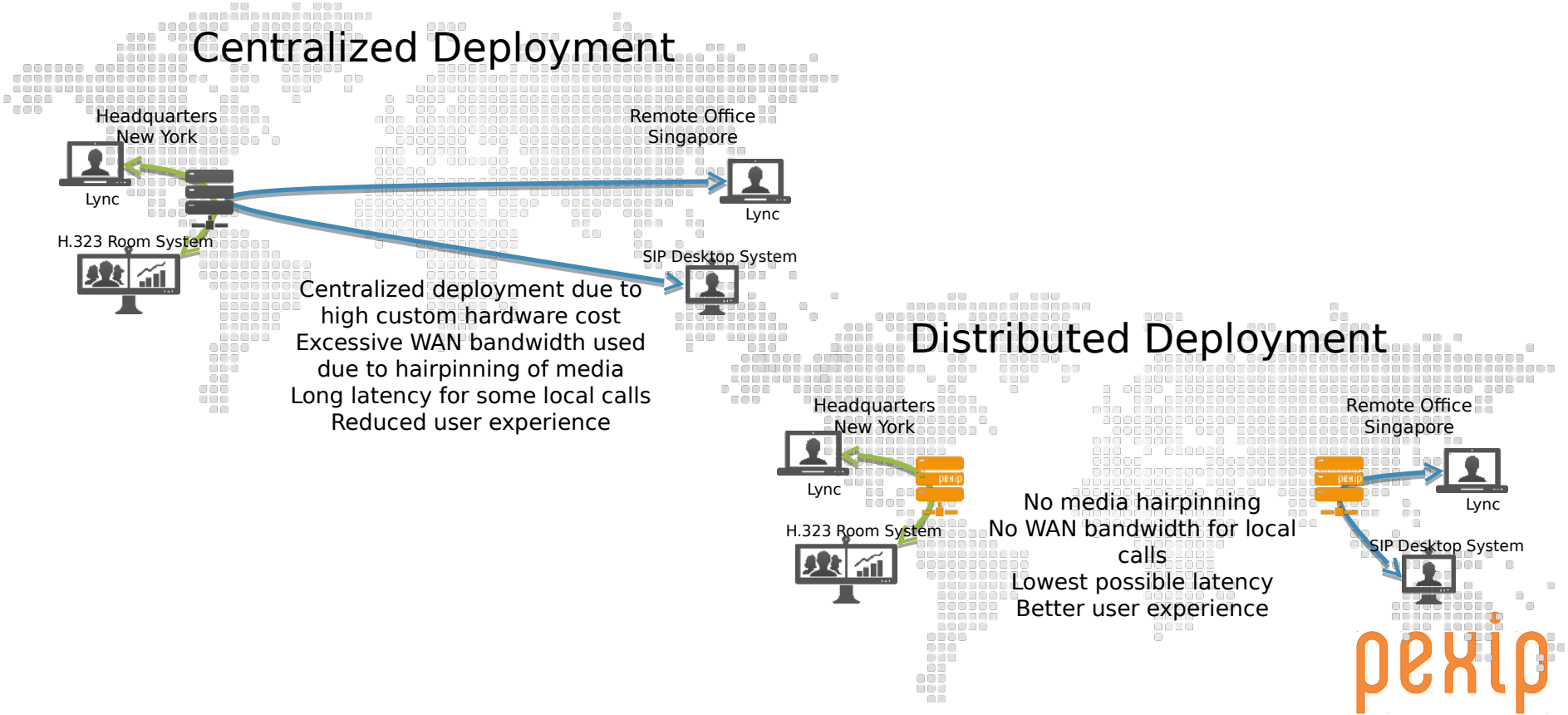


Only one HD stream in each direction
Multiple low bandwidth PIP streams
Resilient to network outages

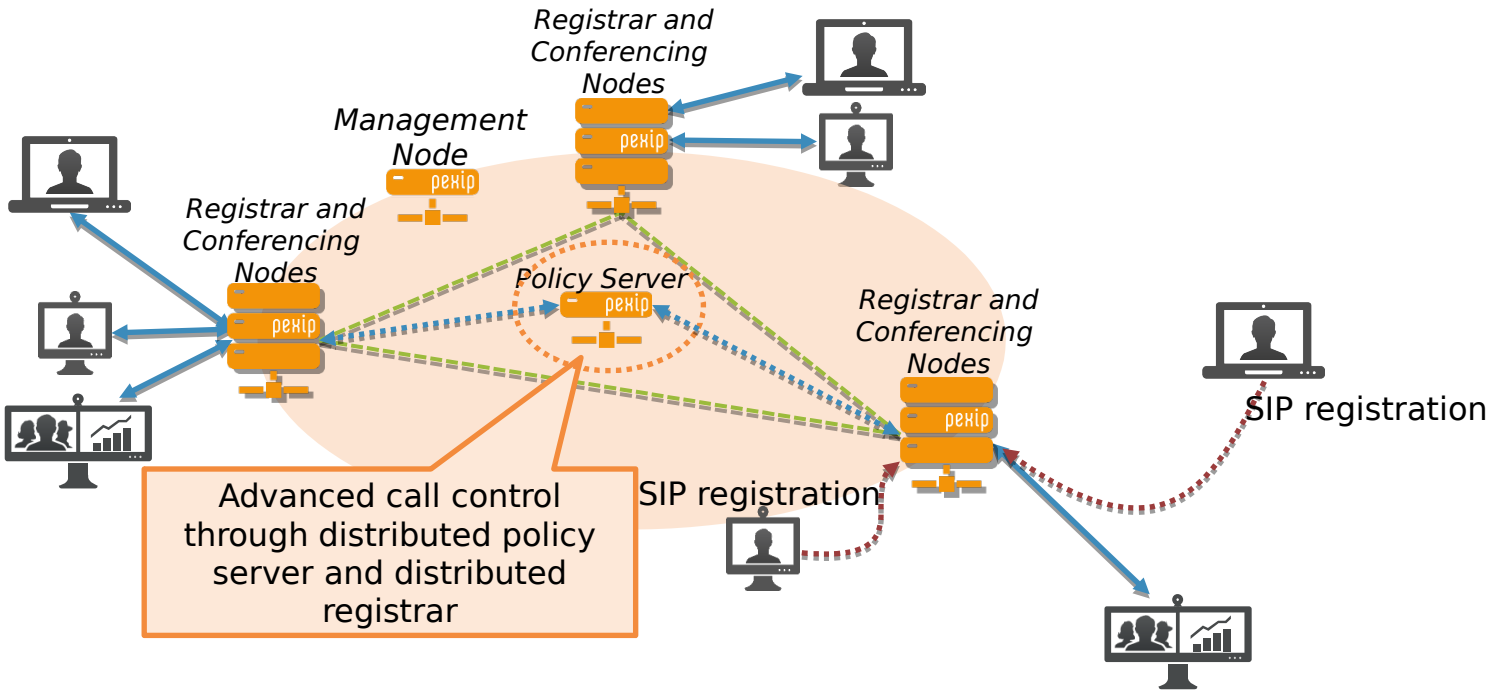
High definition video stream
Low resolution video stream

pexip

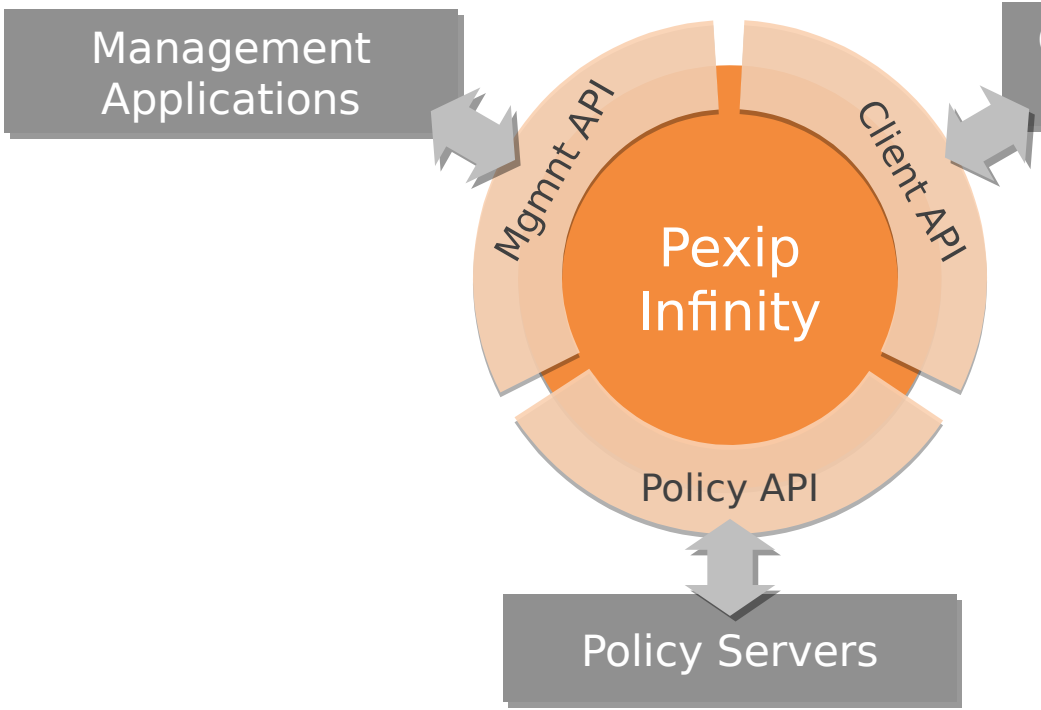
Distributed Architecture - Gateway

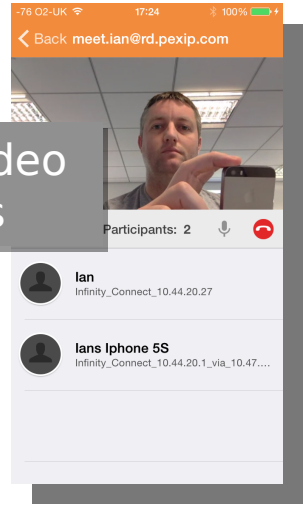


Endpoint registration and call routing

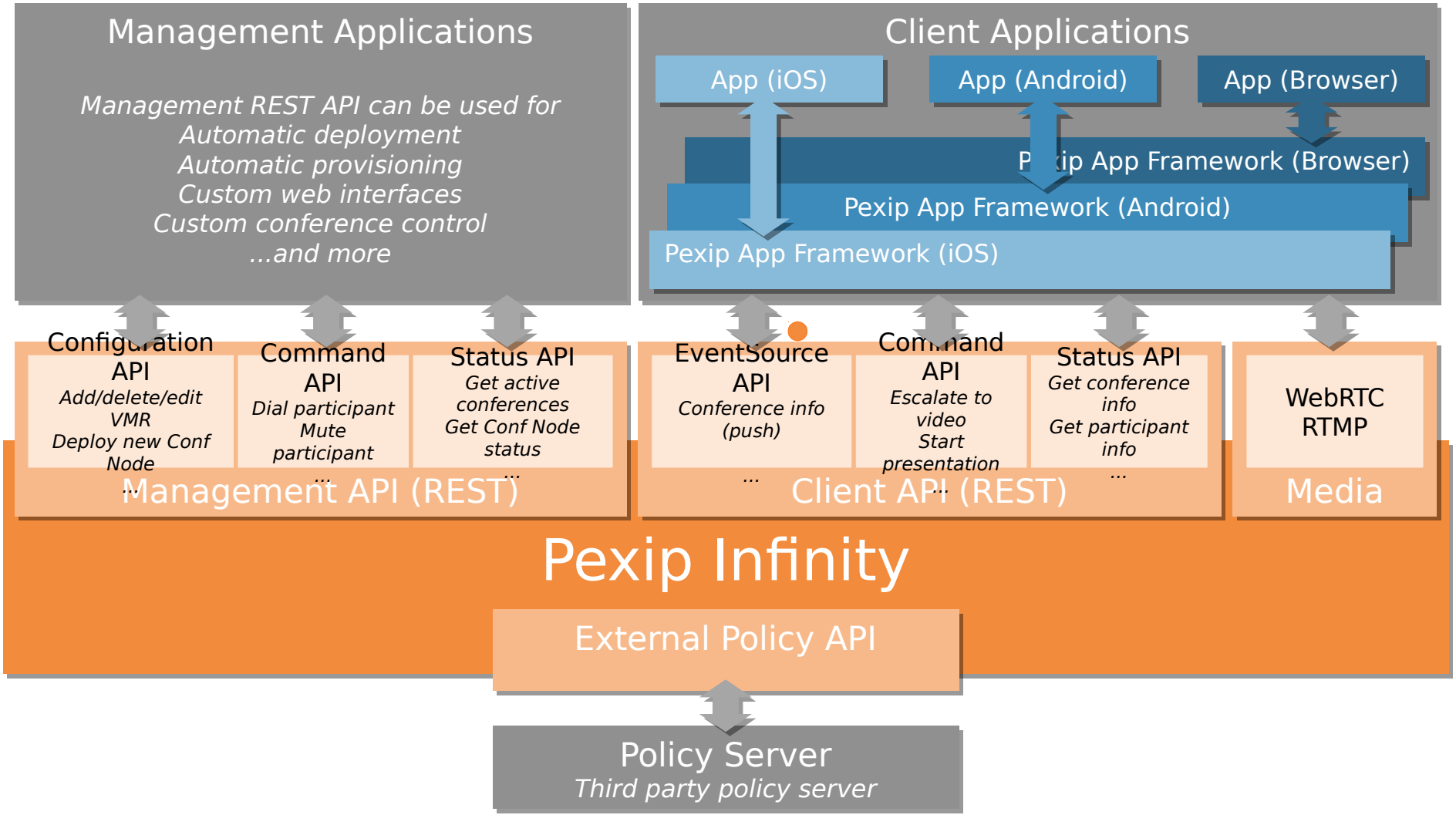


Pexip Infinity APIs in a Nutshell

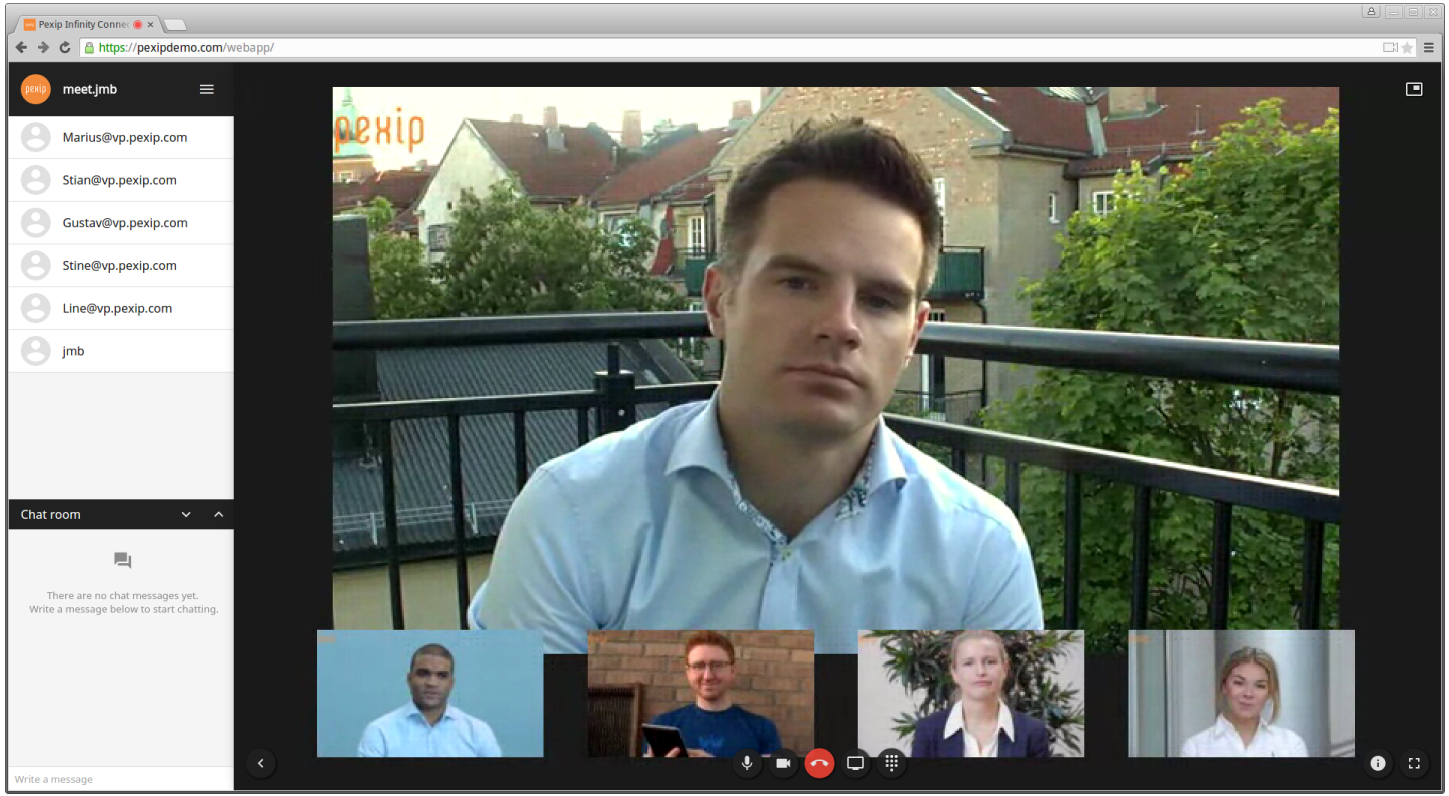




pexip


Management Applications
Management REST API can be used for
Automatic deployment
Automatic provisioning
Custom web interfaces
Custom conference control
...and more
Client Applications
App (iOS)
App (Android)
App (Browser)
Pexip App Framework (Browser)
Pexip App Framework (Android)
Pexip App Framework (iOS)
Configuration API
Add/delete/edit VMR
Deploy new Conf Node
...
Command API
Dial participant
Mute participant
...
Status API
Get active conferences
Get Conf Node status
...
Management API (REST)
EventSource API
Conference info (push)
...
Command API
Escalate to video
Start presentation
...
Status API
Get conference info
Get participant info
...
Client API (REST)
WebRTC
RTMP
Media
Pexip Infinity
External Policy API
Policy Server
Third party policy server

Client API



pexip

Client API overview

- REST, with JSON payloads
- We use it to build our:
  - web application
  - mobile applications for iOS/Android
  - installable client for Windows/Mac/Linux

pexip

Client API flow

- Connect to conference
- (Regularly) refresh connection to conference
- (Optionally) register for event notifications
- (Optionally) activate media functionality
- Disconnect from conference

The same flow works with gateways, too

pexip

Connect to conference

- HTTP POST to
  https://cn/api/client/v2/conferences/conference/request_token

```
import json
import requests
import time

base_url = 'https://10.0.0.1/api/client/v2/conferences/test/'

response = requests.post(base_url + 'request_token',
    headers = { 'Content-Type' : 'application/json' },
    data = json.dumps({ 'display_name' : 'Test' })).json()

token = response['result']['token']
expires = time.time() + int(response['result']['expires']) / 2
participant = response['result']['participant_uuid']
```

Refresh connection to conference

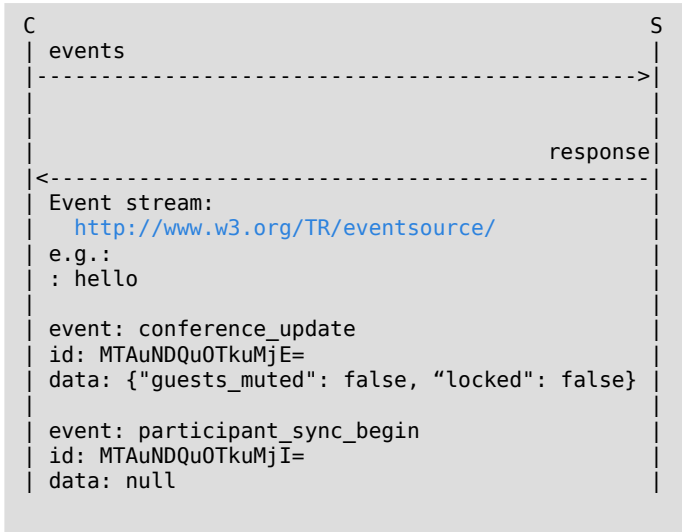
- HTTP POST to
  https://cn/api/client/v2/conferences/conference/refresh_token

```
response = requests.post(base_url + 'refresh_token',
    headers = {
        'Content-Type' : 'application/json',
        'token'        : token
    }).json()

token = response['result']['token']
expires = time.time() + int(response['result']['expires']) / 2
```

Register for event notifications

- HTTP GET to
  https://cn/api/client/v2/conferences/conference/events

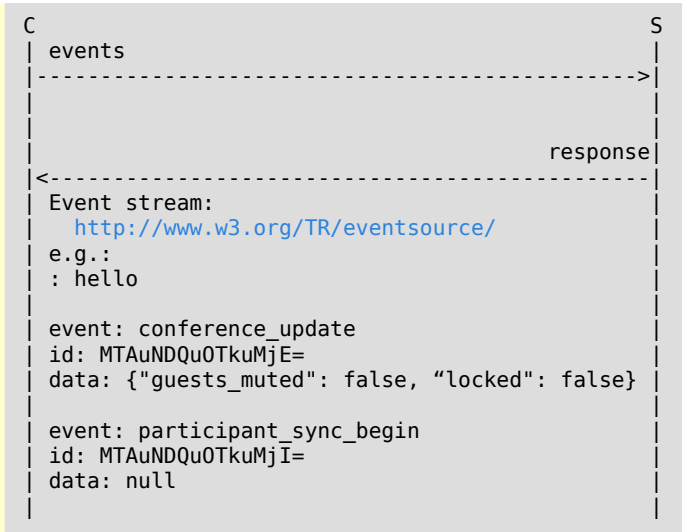
```
response = requests.get(base_url + 'events',
    headers = { 'token'        : token },
    stream = True)

for event in parse_events(response):
    process_event(event)

    if time.time() > expires:
        refresh_token()
```



pexip

Parse the event stream

```
def parse_events(response):
    state = 'BEFORE_EVENT'
    for line in response.iter_lines(chunk_size=1):
        if state == 'BEFORE_EVENT' and line.startswith('event:'):
            state = 'IN_EVENT'
            event_name = line[6:].strip()
        elif state == 'BEFORE_EVENT' and line.startswith(': ping'):
            yield ('PING', None)
        elif state == 'IN_EVENT' and line.startswith('data:'):
            state = 'BEFORE_EVENT'
            event_data = json.loads(line[5:])
            yield (event_name, event_data)

def process_event(event):
    name, data = event

    if name != 'PING':
        # Do something with the event
```



pexip

Types of event

- Participant create/update/delete
  - May be used to display roster list in client
- Presentation start/stop/new slide
- Conference status updated (e.g. lock, guest mute)
- Chat message received
- Disconnected by server

pexip

Activate media functionality

- HTTP POST to

  https://cn/api/client/v2/conferences/conference/participants/participant/calls

```
base_url_p = base_url + 'participants/' + participant  + '/'

response = requests.post(base_url_p + 'calls',
    headers = {
        'Content-Type' : 'application/json',
        'token'        : token
    },
    data = json.dumps({
        'call_type' : 'WEBRTC',
        'sdp'       : local_sdp
    })).json()

call = response['result']['call_uuid']
remote_sdp = response['result']['sdp']
```

Start media flowing (after ICE completes)

- HTTP POST to

  https://cn/api/client/v2/conferences/conference/participants/participant/calls/call/ack

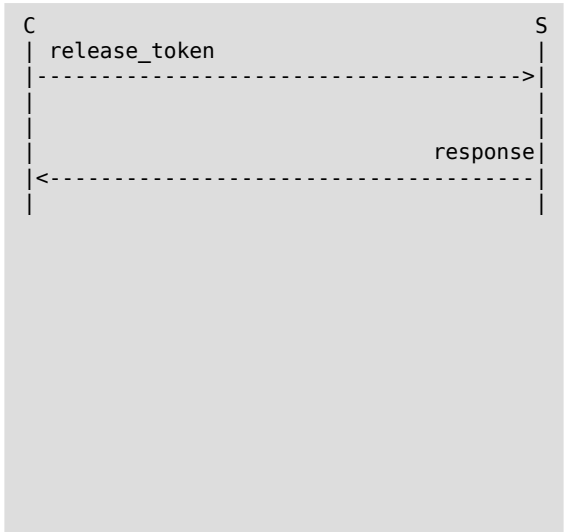
```
base_url_c = base_url_p + 'calls/' + call + '/'

requests.post(base_url_c + 'ack',
    headers = {
        'Content-Type' : 'application/json',
        'token'        : token
    })
```

pexip

Disconnect from conference

- HTTP POST to
  https://cn/api/client/v2/conferences/conference/release_token

```
requests.post(base_url + 'release_token',
    headers = {
        'Content-Type' : 'application/json',
        'token'        : token
    })
```



pexip

Other client API functionality

- Conference control
  - (Un)lock conference (and grant access to individual participants if they join when the conference is locked)
  - (Un)mute all guests
  - Disconnect some/all participants
  - Lock spotlight on a participant, so they are always visible
- Media de-escalation and DTMF entry
- Outbound dialling from conference
- Send chat messages to conference

pexip

Client API forthcoming features

- Presentation sending
  - Allows a client to present a JPEG or PNG image into the conference
  - API is currently beta (web application in Infinity 9 uses it today)
- Client registration, and incoming calls
  - Client may log in to the service and be notified of incoming calls
  - Allows dial-out to WebRTC clients (from conferences, and through gateway rules)

pexip

Client API SDKs

- Javascript
  - Provides a Javascript-friendly abstraction over the client REST API
  - We use it ourselves to build the Infinity web client
  - Designed to be used by third-parties to produce custom UIs and workflows
- IOS
  - Available later in 2015
  - The basis of Infinity Connect for iOS
  - Voice and video capable, in addition to exposing the Infinity client API in a more appropriate way for iOS application developers

pexip

Javascript SDK usage

```
<!DOCTYPE html>
<html>
<head>
  <script type="text/javascript" src="https://cn/static/webrtc/js/pexrtc.js">
  <script type="text.javascript">
    window.onload = function() {
      var rtc = new PexRTC();

      rtc.onSetup = function(url, pin_status) {
          rtc.connect(null);
      };

      rtc.onConnect = function(url) {
          if (url !== null) {
              document.getElementById('remote-video').src = url;
          }
      };

      rtc.makeCall('cn', 'test', 'Test', null);
    };
  </script>
</head>
<body>
  <video width="100%" id="remote-video" autoplay="autoplay">
</body>
</html>
```

pexip

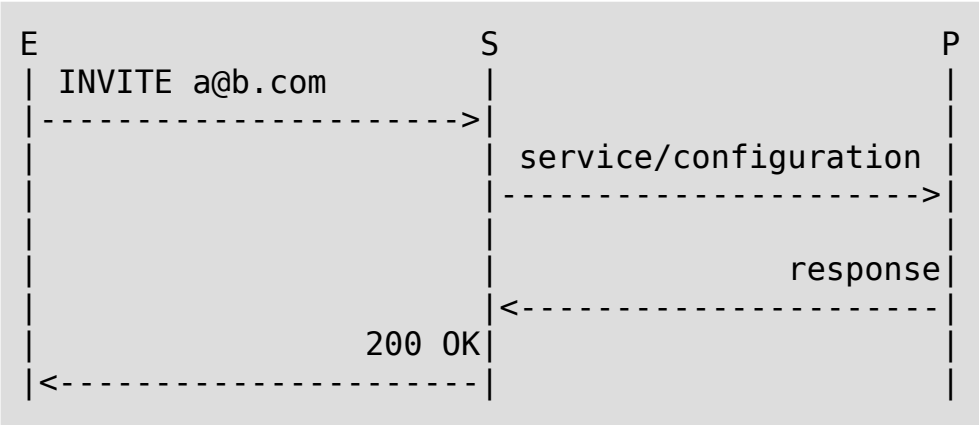
Policy API overview

- New in Infinity version 9
- Simple HTTP(S) GET request/response
- May be authenticated
- Permits external control over (in Infinity v9):
  - Service (VMR/Gateway/IVR) to use
  - Media location
  - Audio avatar
  - Participant avatar (i.e. for display in client roster list)

pexip

Simple policy example

- Goal: allow internal users to connect directly to a conference, but require external users to provide a pin



Service configuration requests

- GET /policy/v1/service/configuration
- Parameters in URL query string:

```
?protocol=sip&node_ip=1.2.3.4&remote_address=4.5.6.7&
local_alias=sip:a@b.com&call_direction=dial_in&encryption=On&
remote_alias=sip:alice@example.com&remote_display_name=Alice&
location=London
```

pexip

Service configuration responses

- Depends on desired behaviour:
  - Send 404 Not Found to fall back to default Infinity behaviour
  - Send 200 OK with Content-Type: application/json to authoritatively define service configuration:
    - Failure: `{ 'status' : 'failed', 'reason' : 'Descriptive reason' }`
    - Success: `{ 'status' : 'success', 'result' : { ... } }`

pexip

Minimal policy response generator

- Assumes web server framework exists already!

```
def create_response(request):
    config = None

    if 'sip:a@b.com' in request.args.get('local_alias', []):
        config = {
            'service_type' : 'conference',
            'name'         : 'test',
            'pin'          : ''
        }
        if not requests.args.get('remote_alias', [''])[0].endswith('@b.com'):
            config['pin'] = '1234'

    return config
```

pexip

Minimal policy response generator

- What if we want to separate guests and hosts as well?

```
def create_response(request):
    config = None

    if 'sip:a@b.com' in request.args.get('local_alias', []):
        config = {
            'service_type' : 'conference',
            'name'         : 'test',
            'pin'          : '1234',
            'allow_guests' : True
        }
        if not requests.args.get('remote_alias', [''])[0].endswith('@b.com'):
            config['guest_pin'] = '9999'

    return config
```

pexip

Other policy applications

- Integrate with Active Directory
  - Dynamically manufacture VMRs based on directory contents and dialed local alias
  - Restrict VMR access to users in specific AD groups
  - Dynamically apply bandwidth restrictions to calls based on importance of participants
- Set conference themes on the fly
  - e.g. on a per-location basis
- Force media to a specific location
  - e.g. to conserve resources
- Custom billing engines
  - e.g. for fine-grained control over multi-tenant environments

pexip


pexip