ALICE: Avaya Labs Innovations Cloud Engagement

John Buford
Avaya Labs Research
Basking Ridge, NJ, USA
buford@avaya.com

Kundan Singh
Avaya Labs Research
Santa Clara, CA, USA
singh173@avaya.com

Venkatesh Krishnaswamy
Avaya Labs Research
Basking Ridge, NJ, USA
venky@avaya.com

ABSTRACT

We present the architecture and implementation of our enterprise cloud portal named *ALICE*, Avaya Labs Innovations Cloud Engagement, which provides self-service access to service developers, tenants, and users to various communication and collaboration applications. Currently ALICE is used for field testing of advanced research prototype services based on technologies such as WebRTC and HTML5. This paper describes the current portal and extensions to support multi-tenancy.

We describe challenges in creating a self-service multi-tenant SaaS (software-as-a-service) portal to host communications and collaboration applications for small to medium scale businesses. The challenges faced and the techniques used in our architecture relate to security, provisioning, management, complexity, cost savings and multi-tenancy, and are applicable and useful to other cloud deployments of diverse enterprise applications.

Categories and Subject Descriptors

H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces – *web-based interaction, computer-supported cooperative work, collaborative computing*; D.2.11 [**Software Engineering**]: Software Architectures – *domain-specific architectures.*

General Terms

Design, Experimentation, Management, Security

Keywords

Cloud, system architecture, portal, multi-tenancy, Internet telephony, enterprise communications, web collaboration.

1. INTRODUCTION

Avaya Labs Innovations Cloud Engagement (ALICE) is a self-service multi-tenant cloud portal which hosts software trials and product releases for communication and collaboration applications for mobile users in small/medium businesses. The self-service portal enables a developer – an internal research team, product group or an external partner – to create and host a service (or application) in ALICE; a customer such as a small business to sign-up to create a new tenant account; and the employees of that business to use one or more approved services. The architecture provides the building blocks for service provisioning, resource isolation, user management and monitoring. It has infrastructure components for communications and collaboration services.

IPTComm'15, October 6-8, 2015, Chicago, IL, USA.

As a portal, ALICE relies on cloud services provided by cloud platforms such as Amazon Web Services, HP Cloud or IBM Cloud Computing. The design of ALICE answers the following questions:

- How can self-service cloud deployment of apps be provided securely?
- What infrastructure is needed on top of existing cloud platforms to support existing and anticipated communication and collaboration services?
- How can the portal enable multi-tenancy and its economies?
- How can provisioning be done so that the scalability is achieved while complexity is avoided?

ALICE aims to host a range of communication, collaboration and analytics services. It contains cloud applications (or SaaS) for both synchronous and asynchronous interactions and sharing but largely focuses on web and mobile-based clients. The diverse applications include both thin-client vs. rich endpoint driven apps, and exhibit many cross application interactions with in-house as well as third-party systems. The scope of this paper is the core architecture of ALICE focusing on multi-tenancy and self-service.

The current services running in ALICE include: (1) a team collaboration system for persistent sharing of content with escalation to real-time voice, video or application sharing [5][6], (2) an enterprise-grade multimedia messaging service [2], (3) a web-based video call and conferencing service that runs all the application logic in the browser [19], (4) a mobile and desktop application to quickly reach one's frequently used contacts, (5) a meeting helper agent which joins the conference bridge during a meeting, automatically creates the meeting summary at the end, and sends it to the participants, and (6) a contact center help desk tool for agents to remotely interact with the customer's browsing session in real-time. Additionally, infrastructure components supporting these services are available for new service developers.

ALICE is positioned within the emerging SaaS (Software as a Service) cloud model. Traditionally, enterprise software is installed within the secure network perimeter of an organization. The SaaS cloud model allows renting such enterprise applications that run in the cloud data centers [9]. Several cloud migration strategies are seen in enterprises: from switching users to a third-party cloud-hosted application (e.g., Google Mail), to creating a parallel cloud-ready application (e.g., Microsoft 365office), to making an existing in-house application cloud-ready (e.g., Avaya IP office).

For vendors, a single migration strategy is often not enough as the underlying architecture often requires interactions with diverse on premise or third-party SaaS systems. This particularly applies to communication and collaboration applications which use infrastructure such as border and PSTN gateways, media servers, relays, transcoders, speech engines, and trunking. For example, a business may want to use cloud-hosted web collaboration and its

in-house VoIP PBX (private branch exchange), and still want both the web and phone users to participate in a conference. Consequently, we believe that the domain of communication services varies in important ways from existing enterprise SaaS offerings. These differences include connection driven workloads which impact resource management and more complex provisioning. ALICE appears to be the first SaaS portal focusing on this class of services while targeting self-service and multi-tenancy.

We present the system architecture of the multi-tenant portal in Section 2, and the operations to enable multi-tenant self-service in Section 3. Section 4 describes our current implementation and the initial set of hosted applications. Section 5 contains application challenges and requirements based on our software development and operations experience. We list related work in Section 6, and present our conclusions and future work in Section 7.

2. MULTI-TENANT PORTAL

Software multi-tenancy refers to a software architecture in which a server running a single instance of a service serves multiple tenants. A tenant is a group of users which shares a security profile with respect to access to the software instance. Each tenant is isolated from all other tenants' data, configuration, user management, resource usage, and tenant-specific individual functionality and properties. Multi-tenancy contrasts with multi-instance architectures, where separate software instances operate on behalf of different tenants [12].

While multi-tenancy increases application complexity, significant resource efficiencies can result. A multi-instance deployment using virtualization means that a complete virtual image including OS and associated services is deployed per tenant. A multi-tenant service would share the OS and services across multiple tenants.

Multi-tenancy impacts ALICE regarding tenant life-cycle, service provisioning, resource isolation, user management, and service branding. Tenant specific configuration, data, resource usage, and user profiles are the province of the service itself. ALICE should support service-specific provisioning mechanisms to enable a new tenant. Each tenant should be able to manage and monitor its suite of ALICE services through a tenant specific user interface.

Multi-tenancy means that different service types and deployments can be simultaneously supported while enabling each tenant to isolate its data and user experience, as if the service were deployed on premise or a private cloud.

This and next sections describe the roles of service developers, portal operator, tenant administrators and end users.

2.1 Service Characteristics

The types of services to be supported are internet collaboration, communications, and analytics, and supporting infrastructure components such as application gateways and transcoders. Such services are conventionally implemented using application servers and the resource load is connection driven as opposed to data driven. The presumed hardware infrastructure is assumed to be that found in commercial and private cloud data centers.

We assume that the services are independent, which eliminates provisioning of service combinations and integrated service management requirements.

This architecture is designed for services which scale towards small/medium business workloads. For more complex provisioning and service management, the service definition provider can provide a per tenant API and web interface.

2.2 New Tenant User Experience

The portal is intended to operate self-service for users, tenant administrators, and service developers. First a tenant administrator registers at the portal. If no tenant is associated with the domain for that user's email address, then the user is routed to the tenant registration page in which tenant info, billing procedures, and licensing is completed. The tenant administrator determines the user workload for each service, and can also moderate user requests to join the tenant group and/or specific services.

A new user self-registers and their email domain is used to determine which tenant group they belong to. The user completes a basic user profile which is available for each service which the user may use. User authentication credentials are created which are used during access to any particular service.

A tenant wants to enable a new service which is listed in the tenant's available service list. The tenant determines the expected user load, geographic placement requirements, and user access constraints. Launching the service causes the ALICE service provisioning interface to be invoked. A new tenant is added to the service, and the expected user load and geographic preferences are used by the service provisioning agent to create the tenant context and add resources according to the expected user load. Per tenant service monitoring is enabled and billing is enabled.

A user selects the service link at the portal to access the service. The link includes the embedded tenant-id and user-id. The service authenticates the user by querying the ALICE user directory (LDAP) server for the combination tenant-id, user-id, and security credentials. Successful query results in access to the service. Within the service, the user's access rights are determined by service specific settings set by the tenant administrator.

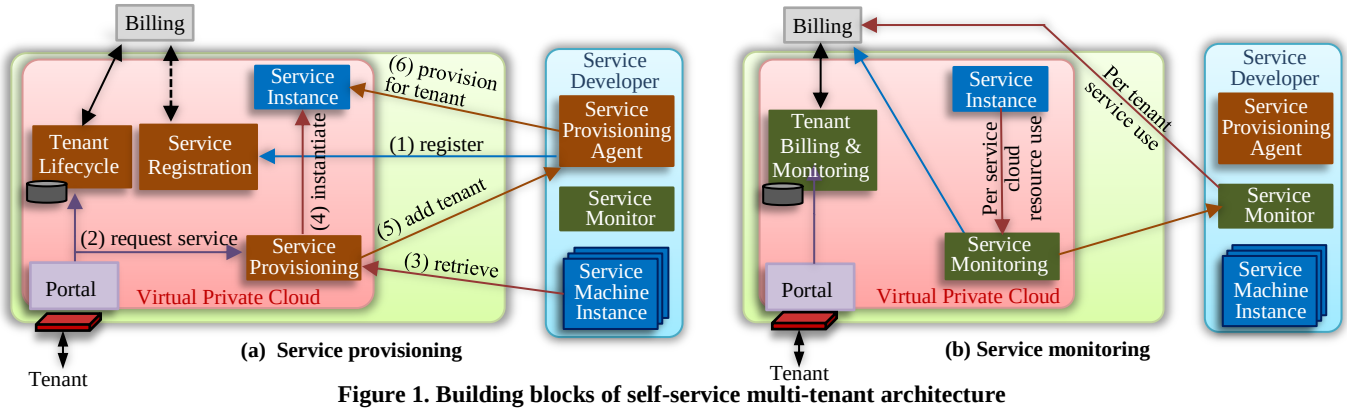

Figure 1. Building blocks of self-service multi-tenant architecture

2.3 New Service Registration

The developer of a new service implements the required interfaces described in the next section and registers the service as shown in Fig.1(a) using the service registration REST API. Registration makes the service available to tenants through the self-service portal. A tenant with access to the service is then able to use wizard style interface to launch a service which users within the tenant group are able to access using the credentials provided by the server. The developer provides one or more machine instances which are to be instantiated when a service is launched. These are stored in the portal service instance catalog and used by the ALICE provisioning interface to launch cloud instances. The developer separately instantiates a provisioning agent which translates provisioning requests from the ALICE API described in the next section to the internal configuration interface defined for the service.

The developer can specify restrictions as to which tenants can use the service. The developer separately establishes a billing profile with the billing system. When the service is launched, the tenant selects from the available billing profiles.

The developer can also specify whether the service developer moderates service access requests, such as for early adopter or trial versions of services.

The developer also specifies the base resource allocated per tenant, and dynamic resource allocation rules by number of users. Redundancy and geographic placement of services are determined by the developer. The service developer determines the mapping of incoming user load to needed resources to sustain the load. For example, IP office service could indicate that one medium CPU instance could support one thousand users. The service developer also determines how the service architecture provides mutual isolation of tenants' data, configuration, security properties, and resource usage. For example, VoIP service could indicate the use of separate tenant database but a shared media relay server.

The service developer is able to monitor resource use for each service. The mapping of resource usage per tenant is determined by the service monitoring interface.

3. SELF-SERVICE OPERATIONS

3.1 Objectives

Multi-tenancy and self-service are at the core of this cloud architecture. The primary objectives are (a) to enable service developers to easily create and host cloud-ready services, and (b) to allow small businesses to easily try out these applications independent of other customers. Unlike existing cloud infrastructure or platforms that bill either service developers or end users, our goal is to price for paying business customers, i.e., tenants.

The architecture focusses on tight security, loose coupling, and separation of customer signup from service access.

1. Tight security: the service must ensure security at different layers – transport, data storage, media path.
2. Loose coupling: inter-service dependencies should be replaceable, e.g., to another third-party or in-house.
3. Separate portal for customer signup: an example of loose coupling is that a service should allow external user signups, e.g., from the portal website.

One of our core objectives is to support bring-your-own-app model, where an existing enterprise application is readily converted to cloud-ready for customer trials, instead of having to re-design the application to fit a particular cloud platform (or PaaS). Without this the experimental cost of application migration for the cloud trial is risky.

To simplify the architecture and encourage quick deployments of services, we make certain assumptions, e.g., keep the services independent of each other, instead of fine-grained-resource-usage-pricing use higher level attributes such as number of users in a tenant group, and delegate several cloud software responsibilities to the service (or SaaS) developers.

A service developer decides how to implement the necessary APIs described earlier, e.g., use true multi-tenant data store or start multiple instances. Although the portal is designed to be multi-tenant, the individual services may not be. Although the portal is hosted on public cloud, an individual service may be on premise.

The following sub-sections describe the interfaces to implement self-service of the multi-tenant model described in the previous section. The tenant uses a web interface to invoke tenant operations.

3.2 Tenant Life Cycle

A new tenant uses the tenant life cycle to register with the portal. This includes selecting services and configuring billing, administration, security and tenant information.

The service selection includes an optional overall non-disclosure agreement (NDA), per service license agreement, deployment options (compute and network resources, geographic, reliability/redundancy settings, and maximum number of users by region.

The tenant billing system is external to the ALICE portal (Fig.1). At a minimum it supports the necessary business processes to establish a new tenant billing account, confirm licensing and NDAs are in place, issue billing statements, and collect per tenant service usage from the service monitoring API.

The tenant creates and maintains the tenant security profile including user management (add/delete/modify); service resource management (add/delete/modify); change tenant information; add and remove services from the service list. Basic tenant information maintained at the portal and controlled by the tenant administrator includes tenant name, domain name, administrator, and branding assets. Branding assets are used by service user interfaces to customize the user experience for that tenant.

The tenant can terminate services and its overall account, including close billing account, delete all user accounts; backup data and configuration; release provisioned instances and resources.

```
class Tenant:
 create(tenantInfo,billingInfo,securityProfile)
 delete()
 modify(properties)
 get():properties
```

3.3 Service Provisioning

ALICE service provisioning interface invokes the service provisioning agent provided by the service developer, passing the tenant information and expected user load. This agent determines the mapping of the user load to the needed resources.

Developers of new services map their service provisioning interface to the following API. The details of how the service creates a multi-tenant instance are isolated from ALICE.

```
class ServiceInstance:
  createForTenant(tenantId)
```

```
start()
stop()
pause()
destroy()
configure()
getServiceSpecificController()
```

3.4 Resource Isolation

Isolation of tenant specific data, configuration, and function are controlled by the service. Resource usage by one tenant should not impact service availability of other tenants. The service developer is expected to ensure data isolation at the storage level by managing separate encryption of tenant databases, and in memory by keeping stateful tenant objects in separate address spaces.

3.5 User Management

User management at the portal level is common profile information across all service instances to avoid duplication of common information. Each service should avoid duplicating collection of this common information.

```
class User:
  addToTenant(userInfo, tenantId)
  removeFromTenant(tenantId)
```

3.6 Monitoring

The tenant, service developer, and portal operator have simultaneous need for monitoring as follows. The portal operator monitors cloud infrastructure usage versus per tenant billing per service as shown in Fig.1(b). The service developer monitors service usage to ensure that resource usage is within the threshold to maintain resource isolation between tenants. The tenant sees usage at the number of users.

The portal operator manages the cloud infrastructure to insure that resource availability is within the agreed upon levels for each service; the service developer determines the redundancy, location, and capacity to provide the per tenant level of service.

4. IMPLEMENTATION

We describe the current implementation containing several real cloud services which have been operating for more than a year. Fig.2 shows the block diagram of the current servers hosted on Amazon cloud (EC2). These are accessed by client applications running in the browser or mobile devices. The portal is available at [1].

Some servers such as for user authentication or VoIP are run inside a virtual private cloud (VPC) and are not directly accessible from outside. Other user facing servers such as the portal website or team spaces are protected behind firewall policies. The system can also utilize third-party cloud services such as for data storage, messaging or media path establishment.

4.1 Enterprise applications

4.1.1 Initial services

Our initial trial has the following user facing services:

1. *Connected spaces*: This is team collaboration software for persistent sharing of content with escalation to real-time voice, video or application sharing [5][6].
2. *Multimedia messaging*: Unified communication and messaging experience delivered via HTML5 technologies, with support for user contacts, directories, messaging, photos and search [2].

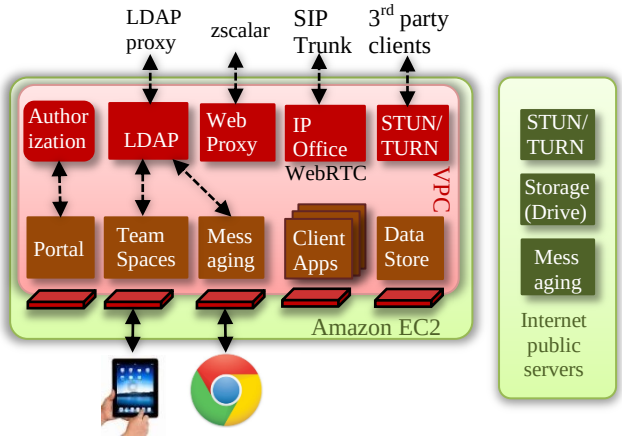

Figure 2. Current implementation contains several cloud services accessible from web or mobile.

3. *Vclick click-to-call*: A pure web-based multi-party video call and conferencing service that runs all the application logic in the browser and integrates with existing corporate directory using a browser extension [19].
4. *Strata mobile app*: A mobile and desktop application to quickly reach one's frequently used contacts, while automatically selecting the right device or third-party application that the target user is reachable on.
5. *Customer-agent co-browsing*: A demonstration of contact center help desk type scenario where a customer interaction can incrementally be escalated to text or voice/video chat with the agent.
6. *Meeting helper app*: This application joins the conference bridge during a meeting, automatically creates the meeting summary at the end, and sends it to the participants.

This is not a complete list of all potential cloud services but is a good sampler of commonly seen communication scenarios.

4.1.2 Role of portal vs. applications

In the current implementation, a new customer signs up on the ALICE portal website (Fig.3) to request access to one or more services. The administrator approves and provisions the services for this signup. Once the user receives an approval email, she can reach one of the approved services from her portal page. The separation of tenant and user management from other services allows us to add or remove services while keeping the list of users intact.

The existing services in ALICE have significant overlap in the technologies and supported use cases. For instance, WebRTC-based communication is part of each of these services. The overlap is intentional because these services are intended to be independent, and solve problems from different perspectives in

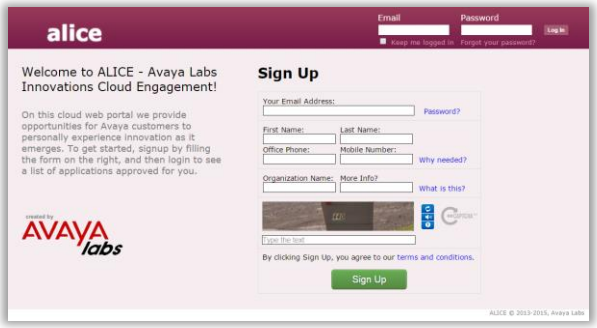

Figure 3. Screenshot of ALICE portal signup page

unified communications and contact center scenarios. Such partly overlapping services are also often seen in the real-world.

Services in our sample set have their own separate databases (mysql, postgresql, sqlite, Cassandra), own separate webservers (Apache httpd, Tomcat, JBoss, or a light-weight Python-based pub/sub server), and are written in many programming languages (Java, PHP, Python, JavaScript). Such diversity represents real-world of communication applications, and is intentional because we do not want to re-implement existing applications to fit a particular cloud platform.

In some cases, we also have cross-service integration. For example, a user on multimedia messaging can attach a message thread to a team space, or people viewing the same document in a team space can initiate an impromptu video conference using Vclick. Wherever possible we repeat the connected service's instance, to keep each service bundle independent of other.

4.1.3 Multi-tenancy vs. multi-instances

Some of our applications such as Vclick and Strata are *endpoint-driven* that run all the application logic in the browser or mobile endpoint. They use a simple WebSocket based resource server for data access and asynchronous events [18]. This resource server is multi-tenant enabled and has a namespace (or tenant-id) attribute to separate the data and interactions among tenants. For example, a Strata user in the first-bank-trial namespace sees a different application branding than the one in the global namespace, and cannot interact with the users in the global namespace.

The team spaces service implements multi-tenancy by using a separate database per tenant. Tenant parameters encoded in the service URL are used to select the database for the given tenant.

Some other services such as multimedia messaging currently use multiple instances to support separate tenants. A front-end reverse proxy can map the URL path or parameter containing the tenant identification to the backend instance. To support self-service multi-tenancy APIs presented earlier, these services may be modified, e.g., to accept a tenant-id and map to a separate database or table per tenant.

4.2 Authentication and user identity

The portal website enables user signup and provisioning. The requirements for user identity and authentication are different for different applications. For example, team spaces and multimedia messaging use unique email as user identity, but Vclick and Strata allow multiple signups from the same email address. Instead, they authenticate using a unique opaque token associated with user's email. Multiple tokens per user enable provisioning different devices of a user with different capabilities.

The portal website uses LDAP to create user signups, and a MySQL database to store list of approved apps. Beyond that individual service implements its own authentication using either LDAP or a proxy to map user-id to opaque auth-token.

It is tempting to add more features to the portal such as for user profile and picture, which could be reused by other services. However, not all services have the same set of social profile requirements. It is also tempting to create a single-sign-on (SSO) that could be used by any service. However, that will break the diversity requirement, forcing all services to adhere to a specific way of doing things. For example, a service may want to link a user to her enterprise identity bypassing this SSO. Keeping the authentication in each service gives more flexibility to the developer on how to implement it.

4.3 Multi-Tenant Provisioning

The ALICE service provisioning and monitoring is intended to be independent of the underlying cloud platform. For example, Table 1 shows a top level mapping of the service registration, provisioning, and monitoring for ALICE to AWS [4] and OpenStack [16]. An authorized developer is given credentials to perform operations on its cloud services that are hosted in ALICE.

Table 1 Cloud platform operations

ALICE	*AWS EC2 and CloudWatch API*	*OpenStack*
Service register	ImportImage RegisterInstance	CreateImage RegisterImage
Service provisioning	StartInstances RunInstances StopInstances	Create Server Resize Server Start Server Stop Server
Monitor service	MonitorInstance ListMetrics CPUUtilization NetworkIn/Out	Show Resource Info Create Meter Show Meter Statistics

5. CHALLENGES AND REQUIREMENTS

This section lists some practical challenges faced and techniques used in implementing and deploying individual services. At the high level, there are four types of the challenges in enterprise cloud software: (1) compliance/security, (2) cost or price, (3) complexity, and (4) compatibility with existing IT systems.

5.1 Compliance and security

Enterprises often have strict compliance requirements, especially for confidential data and real-time voice interactions. Multi-national organizations also comply with inter-country data sharing laws. Telecom regulations and requirements related to accounting, auditing, call recording and data privacy further complicate existing enterprise communication systems. A customer would like to export and delete all its private data stored on ALICE when the trial is discontinued or completed.

5.1.1 Private cloud

Amazon VPC (virtual private cloud) gives the necessary tools to configure a secure and private data center in the cloud for our system. In particular, security critical pieces such as user authentication, media gateways and transcoders run in a VPC isolated from the rest of the Internet. Additionally, encrypted IPsec tunnels can be established between the VPC and the customer's data center when desired.

5.1.2 Transport security

Transport layer security (TLS) is used in all client-server and server-to-server interactions for data exchange, signaling or control. We require all web accesses over https, and block any unsecured http access except for the front-page, which when accessed on http is redirected to https.

Individual services such as team spaces and multimedia messaging also require client certificate from the browser to restrict unauthorized clients at the transport layer. When accessing these services, the browser prompts the user to select an installed certificate (or user profile) on both mobile and desktop. Per-tenant or per-user certificate can provide fine grained access control.

Although many existing services can terminate TLS, it is not always possible, e.g., if secure transport or client certificate is not (or is incorrectly) implemented in the server, or a self-signed

instead of a PKI server certificate is used. In that case, we use nginx or stunnel proxy to terminate TLS and reach the backend insecure server in the VPC. Using a proxy shields the client-server security requirements from the capabilities of the server.

5.1.3 Cross origin resource access (CORS)

Web browsers prevent cross-origin Ajax connections to avoid leaking service APIs to unsolicited third-party websites. However, such cross-origin access is sometimes required, e.g., when the pub/sub event server runs on a different port than the application server, or in cross-service interaction.

Our WebSocket pub/sub server allows only white-listed Origin. Some others services do not allow CORS, or incorrectly return Access-Control-Allow-Origin: * header, which does not work for Ajax. In that case, we use nginx reverse proxy to terminate and approve legitimate CORS using more_set_headers function.

Browser prompts the user for selecting client certificate or for continuing on an invalid server certificate – only if the origin's webpage can be displayed, but not for cross-origin Ajax or WebSocket, in which case it silently fails the request. One could use nginx, e.g., by forwarding /pubsub to the external server, and all the other /* to the local backend server. In team spaces, we use a nested iframe in the first webpage that includes a blank page from the second pub/sub server. Since the iframe is displayable, the browser shows the prompt and succeeds the connection.

Browsers disallow loading insecure http content within an iframe of a secure https webpage. Team spaces hosted on https may need to show third-party http content in its document sharing tab, which is overlaid with application code for annotations and interactions. We use a data URL to encapsulate a subset of the application code that loads the http content and enables annotations. This URL is loaded in a new tab. The client-server application remains secure on the network; only the new tab is insecure but not on the network. Alternative approaches have problems, e.g., launching the document in a new tab loses the application code, degrading the service to http loses the session security and privacy, and using a server-side proxy on https to serve third-party http requires inefficient substitution of internal links via the proxy.

5.1.4 Access control

Our services typically do user-level access control using web style username/password and cookies. HTML5 local storage is preferred over cookies to store certain sensitive credentials.

The resource server uses an opaque authentication token (can be many per user) that is transparently passed from the browser's local storage during connection establishment. It allows fine grained resource authorization, e.g., Alice but not Bob can POST to /users/alice@example.net/presence, and avoids same password reuse that often causes accidental leak to another website.

5.1.5 Employee directory

Enterprises use internal directories, e.g., LDAP, which must remain private and not moved to the cloud. We have created a split-proxy architecture (Fig.4) that enables an authorized Internet application to access intranet services such as LDAP or internal web pages. The split-proxy terminates the server connection at A from the co-located website acting as a client, securely tunnels the connection request and subsequent data to its intranet component B, which re-originates the connection and request in the enterprise network. The response traverses back to the client. The split-proxy is protocol independent, i.e., it does not parse LDAP.

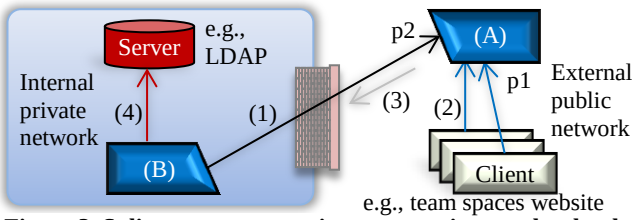

Figure 3. Split-proxy to access intranet services on the cloud

5.1.6 Firewall filtering

Amazon EC2 firewall filters are used extensively to block any unwanted ports, especially for VoIP, e.g., block inbound SIP from the Internet.

Although using only public-key instead of password-based login mitigates malicious ssh attempts, the crucial log files get filled with garbage, which could potentially hide a real problem. We use iptables filter to prevent immediate ssh attempts from the same IP address, thus reducing the frequency of such attempts in the log.

5.1.7 Miscellaneous

Our services use WebRTC media paths encrypted using DTLS-SRTP in browser-to-browser as well as client-server. We use HTTPS/TLS for all signaling and control data. This ensures secure media path as long as the browser and server are not compromised.

We use CAPTCHA during signup at the portal website. It is also recommended if frequent failed user input attempts from the same source are detected on any web page.

5.2 Complexity

The complexity in enterprise cloud systems is multi-faceted. It is risky to get locked to a single cloud platform, and complex to move to a different one later. Many services can readily be scaled horizontally, e.g., by adding more instances of the media relay or partitioned databases. However, vertical scaling and performance improvement in terms of CPU and memory utilization is harder.

Software development and deployment on the cloud often involve extended troubleshooting, especially for communication systems that must deal with NAT and firewall traversal, corporate web-proxy or VPN interference, flaky Internet connection, or race conditions due to signaling or media path latency. We show some steps in reducing software development/deployment complexity.

5.2.1 Local development

Our endpoint driven applications are developed and tested on local host using a command line Python-based web server. Since all the application logic runs in the browser, this significantly reduces the development time. Once tested, only one line is changed to switch to the cloud pub/sub server.

We use multiple user profiles on locally running browsers to emulate multiple user accounts.

Our communication and collaboration services allow multiple registrations from the same user, and the ability to join the same conference as multiple participants by the same user. This greatly simplifies testing of collaboration applications from one machine.

In the past, some of our services such as team spaces and web collaboration have been fully hosted on a single laptop for demonstrations or exhibitions with limited access to Internet.

5.2.2 Corporate firewall and web proxy

Old style web proxies often interfere with modern WebSocket connections, either during handshake preventing upgrade, or

during long lived session by disconnecting TCP after a timeout. The latter is solved by application-level keep-alive messages. The former usually requires secure WebSocket over https to make the handshake opaque to the proxy.

In some networks outbound ports are blocked except for https. We use nginx reverse proxy on 443 to multiplex many services, e.g., path /cs/app goes to team spaces and /restserver to resource server.

Corporate firewalls usually block peer-to-peer media flows of WebRTC. We use a cloud hosted media relay to facilitate media paths across such restricted network edges. Some services such as IP office force a client-server media path for all calls to enable advanced media processing at the server.

5.2.3 NAT'ed servers

Amazon EC2 has popularized the concept of running servers behind a NAT. This is not an issue for traditional web services. However, some VoIP systems send IP addresses in the application call setup messages, and these servers must be made aware of the external IP addresses via configuration for use in any application protocol. Alternatives such as STUN-like mechanism at the server or address substitution at a proxy may not work in practice because not all strings that resemble an IP address should be substituted, substitution may not always be feasible due to obscure binary format, such substitutions may break the message digest or signature, or different IP addresses may be used in forward and reverse web traffic due to reverse proxies.

5.3 IT compatibility

Enterprise software is often built and tested for specific platform, and at times, even a specific OS version. Sometimes, such software is installed as a complete bundle including the OS on the target machine. Often times cloud innovations of resource sharing cannot easily be applied when backward compatibility with such configuration is required.

Enterprise software sometimes depends on crucial pieces of external services that are readily available in the enterprise, but not in the cloud, e.g., employee directory for accounts verification or search, and exchange server for user's calendar to join the next meeting. Fig.3 shows one approach to solve this. More generally, it is important for a tenant to preserve its own authentication information in ALICE that it would use in its own domain.

5.4 Cost

Saving cost is usually one of the main objectives of migration to cloud. However, SaaS cost saving requires careful optimization as well as long term vision. Risks such as accidental configuration error in auto-scaling, hidden cost of periodic or frequent required upgrades, or double charging for bandwidth in the cloud as well as enterprise's broadband are crucial. For instance, a cloud provider may force an upgrade to Java 7 which may require significant effort in refactoring the application software written in Java 6.

In our case, the cost also includes the development and deployment efforts for new services to make them cloud ready for customer trials. Such hidden or intangible costs are hard to measure or estimate.

The measurable operational cost includes the cloud's monthly bill, and annual cost for domain names and SSL certificates. Hosting multiple server applications on the same EC2 instance, picking the right instance for experimental projects, and stopping instances when not in use are all useful in saving the monthly fee. The SSL certificate cost can be reduced by running multiple servers on different ports on the same IP address because the server certificate bound to an IP address works for any port. The nginx reverse proxy can also reduce the number of server certificates needed, by terminating https accesses to all the services on one or two front-end proxy machines.

5.5 Service developer guidelines

We list some guidelines for service developers that are desired but not required to host the services on ALICE.

1. Decouple user identity and authentication from rest of the business logic, so that user signup and in future user authentication can be moved outside the service.
2. Loose coupling among multiple components within a service, or with any external service, so that one can be easily replaced or moved.
3. Avoid use of locally detected IP addresses in server's application protocol. Prefer to use configuration.
4. Use tenant-id in any data access or server APIs, so that it can be easily made multi-tenant in the future. Ability to dynamically attach a database server or select a database based on this tenant-id on a per-request basis.
5. Do not assume specific transport or network layer security or access control, as some of these can be moved to outside the application server via a reverse proxy.

6. RELATED WORK

6.1 Self-Service SaaS

Microsoft Office 365 [13] is an example of SaaS group of software plus services subscriptions that provides productivity software and related services to its subscribers. Details of how services are auto-provisioned or monitored on Microsoft cloud infrastructure are not available.

Mietzner and Leymann [14] present a self-service portal for deploying multi-tenant services. Services are described by templates which contain an application model and a variability model. The service vendor develops the application model and uploads it to the portal with the solution components. The customer uses the portal GUI to select app-specific variability parameters; the details of this configuration step is specific to each service.

ALICE differs in two ways: 1) it delegates the provisioning steps to a provisioning agent that the developer would provide for their service, 2) it collects common provisioning parameters, e.g., user load, and delegates the resource assignment to service developer.

6.2 Cloud Platforms

Several cloud platform APIs are available, including AWS [4], OpenStack [16], vCloud [20], and Cloud Foundry [7]. These platforms are not intended for end-user self-service access to services, but provide infrastructure to streamline the development, deployment, and management of cloud services.

Amazon Web Services [4] (AWS) is a collection of remote computing services which make up a cloud-computing platform. Instances can be currently launched in 9 regions worldwide. Developers can launch instances from their own machine images or select from a large catalog of third party images. OpenStack [16] is a free and open-source cloud-computing software platform. Users deploy it as an infrastructure-as-a-service (IaaS). It can be used to control pools of processing, storage, and networking resources—which users manage through a web-based dashboard, through command-line tools, or through a RESTful API. The VMware vCloud API [20] provides support for developers who are building interactive clients of VMware vCloud Director using a RESTful application development style. Cloud Foundry [7] is an open source cloud computing platform as a service (PaaS).

Applications deployed to Cloud Foundry access external resources via services. External dependencies such as databases, messaging or file systems are services. When an application is pushed to Cloud Foundry, the services it uses are specified.

Unlike existing PaaS environments that require changing the software application to fit the desired model, ALICE leaves it up to the service developer, who may replicate the server instance on a new machine (or VM) for the new tenant, attach a separate database (or table) on-demand per request based on tenant-id, partition the data in same database based on tenant-id, and/or provide access control based on tenant-id.

6.3 SaaS Multi-Tenancy

Previous work in SaaS multi-tenancy includes platforms, database impact, and design of specific services. Krebs et al. [12] provide a succinct definition of multi-tenancy and the architecture issues for designing multi-tenant services. Force.com [17] appears to be the largest deployed multi-tenant SaaS platform. All application data is treated as meta-data to the Force.com platform, which is a Development as a Service (DaaS) paradigm.

Jacobs and Aulbach [10] describe design alternatives for multi-tenant databases. They examine three approaches: database per tenant, separate tables per tenant, and separate rows in each table per tenant. Separate databases give the best isolation but have the most overhead. Table sharing complicates isolation, query optimization, and tenant data backup. Walraven et al [21] describe a middleware technique for performance isolation in multi-tenant SaaS architectures. The middleware handles scheduling and monitoring according to SLA constraints.

7. CONCLUSIONS AND FUTURE WORK

The goal of ALICE is to accelerate and streamline the deployment of communication and collaboration services hosted in the cloud to both end users and small/medium businesses. Self-service and multi-tenancy are key to achieving this. The design addresses self-service for tenants, users, and service developers. The model is practical because of availability of versatile cloud platforms and assumptions of service independence and management that we believe are realistic for the range and complexity of services delivered to small and medium size businesses.

There are known limitations in dynamic resource management on these platforms. For example, our experiments with Amazon EC2 show that horizontal scaling is feasible whereas automated vertical scaling is not yet. For operating a scalable multi-tenant service, the lack of vertical scaling is less than ideal but not insurmountable. Other issues include how a tenant would coordinate identity management used in ALICE with other SaaS portals and with its own enterprise directory.

The design of a multi-tenant service currently requires service-specific architecture. Application servers available today do not include pre-defined support for tenant data and resource isolation. The identification of generic multi-tenant building blocks for communications and collaboration services is an open question.

Individual components used in multimedia communication and Internet telephony may have different multi-tenancy requirements than traditional data-oriented services, e.g., resource consumption, isolation or privacy of a media relay is dealt differently than a data storage service. Self-service multi-tenancy of such components is for further study, especially in hybrid cloud architecture.

8. REFERENCES

[1] ALICE: Avaya Labs Innovations Cloud Engagement. https://alice.avayalabs.com. Website. Retrieved Jul 2015.

[2] Avaya Multimedia Messaging. http://www.avaya.com/usa/documents/avaya-multimedia-messaging-uc7657.pdf. Ret. Jul 2015.

[3] Amazon Virtual Private Cloud (VPC). Website. http://aws.amazon.com/vpc. Retrieved Jul 2015.

[4] Amazon Web Services API. http://docs.aws.amazon.com /AWSEC2/latest/APIReference/Welcome.html. Jul 2015.

[5] Buford, J., Dhara, K., Krishnaswamy, V., Wu, X., Kolberg, M. A Communications-Enabled Collaboration Platform: Framework, Features, and Feature Interactions. *Principles, systems and applications of IP telecommunications*, IPTComm. New York, NY. 2010.

[6] Buford, J., Mahajan, K., Krishnaswamy, V. Federated Enterprise and Cloud-based Collaboration Services. *IEEE Intl Conf on multimedia system architectures and applications*, IMSAA-11. Bangalore, India. 2011

[7] Cloud Foundry. http://cloudfoundry.org/. Ret. Jul 2015.

[8] Chieu, T.C., Mohindra, A., Karve, A.A., Segal, A. Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment. *IEEE Intl Conf on e-Business Engineering*, ICEBE'09. pp.281-286, 2009

[9] Enterprise software vs software-as-a-service. http://effectivedatabase.com/resources/the-difference-between-enterprise-software-and-software-as-a-service/. Retrieved Jul 2015.

[10] Jacobs, D., and Aulbach, S. Ruminations on Multi-Tenant Databases. In BTW, vol. 103, pp. 514-521. 2007.

[11] Katzer, M., Crawford, D. Office 365 DirSync, ADFS, Single Sign On and Exchange Federation. 2013.

[12] Krebs, R., Momm, C. and Kounev, S. Architectural Concerns in Multi-tenant SaaS Applications. *Proc. of the 2nd Intl Conf on Cloud Computing and Services Science (CLOSER 2012)*. 426-431. Portugal, 2012.

[13] Microsoft Corp. *Office 365*. http://office.microsoft.com/. Website. Retrieved Jul 2015.

[14] Mietzner, R., and Leymann, F. A self-service portal for service-based applications. *2010 IEEE Intl Conf on Service-Oriented Computing and Applications (SOCA)*, pp. 1-8.

[15] Nginx. http://nginx.com/. Retrieved Jul 2015.

[16] Open stack API. http://developer.openstack.org/api-ref.html. Retrieved Jul 2015.

[17] Salesforce.com. Force.com: A Comprehensive Look at the World's Premier Cloud-Computing Platform. 2009. http://www.developerforce.com/media/Forcedotcom_Whitepaper/WP_Forcedotcom-InDepth_040709_WEB.pdf

[18] Singh, K., and Krishnaswamy, V. Building communicating web applications leveraging endpoints and cloud resource service. *IEEE Intl Conf on cloud computing* (IEEE Cloud). Santa Clara, CA, USA. 2013.

[19] Singh, K., Yoakum, J. and Johnston, A. Enterprise WebRTC powered by browser extensions. *Principles, systems and applications of IP telecommunications*, IPTComm. Chicago, IL. 2015.

[20] VMWare vCloud Air. http://vcloud.vmware.com. Jul 2015.

[21] Walraven, S., Monheim, T., Truyen, E. and Joosen, W. Towards performance isolation in multi-tenant SaaS applications. *In Proc. of the 7th Workshop on Middleware for Next Generation Internet Computing*, p. 6. ACM. 2012