Paper title: ALICE - Avaya Labs Innovations Cloud Engagement.
Speaker: Kundan Singh. Track: IPTComm.

Hello everyone. My name is Kundan Singh. This presentation is about ALICE, or Avaya Labs Innovations Cloud Engagement.

Story of ALICE

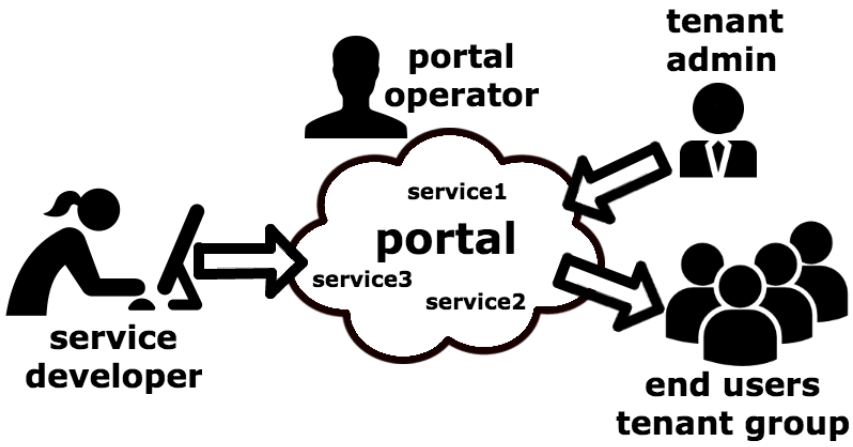
Avaya Labs Innovations Cloud Engagement

A long time ago, Avaya spun out of Lucent to innovate and focus on enterprise communication systems. These systems were and are installed on premises in customers locations or their data centers.

Over the years, with growing number of communication offerings, on site maintenance became a problem. Thankfully, the emergence of cloud gave a new hope.

This is our story in Avaya Labs, of creating a cloud portal to allow our developers to create services and our customers to try them out...

The basic idea is shown here.



A service developer on the left could be a small team within Avaya or an external partner, who creates some application or service, such as a VoIP or messaging system. The tenant is a small or medium business on the right who registers for trying out one or more services. The tenant admin enables some services for its users, and the end users in that tenant group use the service.

There are multiple services developers, and multiple tenants. The portal shields the developers from the tenants, so that the services and tenants can come and go in any order for the trial.

This model is particularly useful for many research prototypes and early enterprise systems that are created as an enterprise software but need a cloud trial with existing customers before deciding to make it a product.

This is a list of currently hosted services in ALICE.

Current services

1. Team spaces
2. Multimedia messaging
3. Vclick click-to-call
4. Strata mobile app
5. Customer-agent co-browsing
6. Meeting helper app

Team spaces or connected spaces is a team collaboration system for persistent sharing of content among team members, and for escalation to real-time voice, video or application sharing. It also has the ability to share documents, meeting notes, etc., among team members.

Multimedia messaging or Avaya Multimedia Messaging (AMM) service enables unified communication and messaging using HTML5 technologies, and has support for user contacts, directories, messaging, photos and search.

Vclick and Strata are endpoint driven apps that use a light weight resource service in the cloud. Vclick is a pure web based video call and conferencing application, that can integrate seamlessly with existing exterprise systems such as corporate directory or telephony gateway.

Strata mobile and desktop app allows quickly reaching yourfrequently used contact on whatever device or app the contact is available on. Although it is targeted for mobile devices, there is an equivalent desktop app as well.

Then there is a demonstration of contact center help desk, where a customer service representative can use co-browsing with

real-time text or voice/video to help a customer walk through web navigation and form filling.

The meeting helper app joins a conference bridge on behalf of the user during a meeting, captures meeting notes and meta data, and sends the meeting summary at the end to all the participants.

In this video demonstration, you will see the current implementation of the portal. And I will also show multi-tenancy for a few services.

Click to see a demo video

The user named Green logs on the portal website. List of approved services (or apps) are shown. Also shown are the the basic user profile on the portal, the additional tenant groups besides the default one derived from the user's email domain, and other users in that email domain who have signed up.

Let me expand one of the services. The team spaces service page can be launched from here for desktop, as well as for mobile. The service specific user profile is editable on the specific team spaces service website.

This user belongs to additional tenant group of first bank. So if the user selects that group, then opening the service page takes her to the customized tenant specific connected spaces instance. Similarly for the mobile website as you can see here.

Strata is another service shown here. Two instances of the Strata app on desktop are shown. User henley logs in on the left application and sees his top 9 contacts. Then he uses the secure token obtained from a different tenant group of First Hospital customer trial, and logs in on the right side application.

Here you can see both apps are logged in with the same portal user account, but in different tenant groups. The Strata app uses data partitioning for multi-tenancy, as you can see with customized look-and-feel and different set of top 9 contact data for the same user, even though the same service and same client app is used.

This user is logged in as an admin. The admin interface shows the list of users signed up, and their non-default services enabled, and their non-default tenant groups assigned. The admin can modify these attributes for individual users.

Rest of the presentation is going to describe in more detail about this system. In particular, first part of the paper focusses on the self-service and multi-tenancy architecture in the portal. And how that is different from what's out there.

And the second part deals with the individual services in our implementation. We also show some tips and tricks to solve some non-trivial problems encountered based on our practical experience.

The portal deals with tenant life-cycle, when tenant joins or leaves the trial, provisioning of individual services, resource isolation between services, and user management independent of individual services.

The service developer can create tenant specific config or data, monitoring and billing. It can also require service specific user profile beyond the basic profile needed by the portal.

A service developer, which is a small team or external partner, creates and registers a service with the portal. It provides information about the instances to be used with the service, e.g., a VoIP system could include machine instances for a web server and a media relay.

It maps the provisioning API of the portal, e.g., when a new VoIP instance needs to be launched, it could launch the specific machine instances for the web server and media relay, if needed.

The developer can create service specific billing profiles, and define tenant specific access control. It can define resource usage limits per tenant. And resource rules, e.g., say the media relay can handle 1000 users, and if a tenant has more users, then create more instances of the relay.

The developer can also define resource redundancy for different components of the system, e.g., use separate web server and data base for different tenants but share the media relay if needed.

A user can signup as a tenant adiminstrator. By default the first email in a domain becomes the admin for that domain's tenant group. The tenant admin provides billing information, e.g., credit card, so that all the resource usage of the tenant group is billed to the tenant.

The admin essentially represents the customer, and signs any licensing or NDA. It can moderate the users permissions within that tenant group. Finally it can launch individual services for its tenant group and enable monitoring or billing for its users.

A regular user, during signup, fills the initial user profile on the portal, and later service specific user profile as needed. He can visit the service link or app from the portal website as I showed in the demo.

Next, I will talk about, in further detail, some of the self-service and multi-tenancy aspects of the portal architecture.

The paper describe the necessary building blocks to enable self-service multi-tenancy. It identifies various interfaces for service provisioning and monitoring. One thing to note is that billing is kept outside the purview of service developer or portal, but necessary interfaces enable the billing mobile to get the resource usage data.

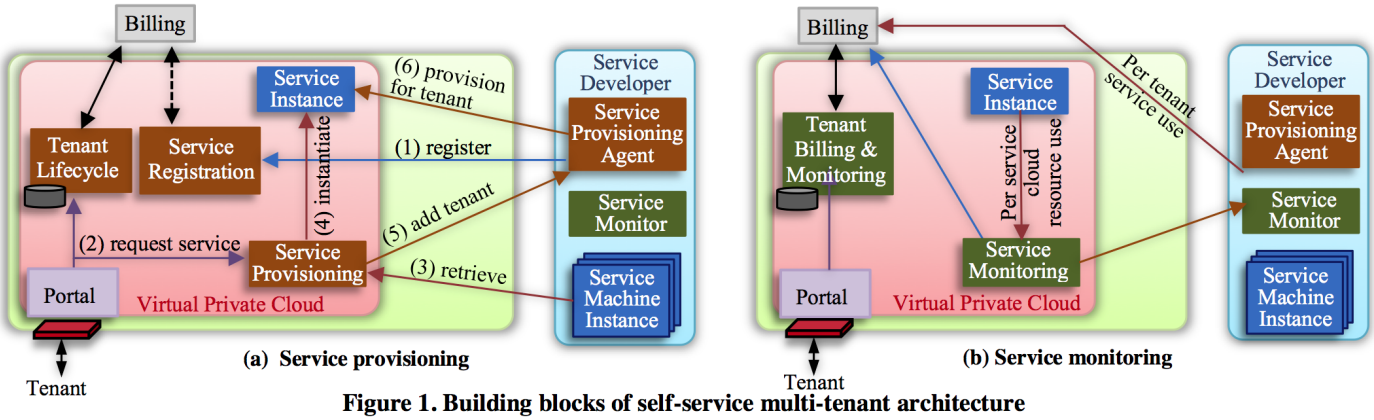


Figure 1. Building blocks of self-service multi-tenant architecture

---

Here I will give an high level list of what the paper describes, but if you are interested in how, please read the paper.

Paper describes...

How does multi-tenancy impact the portal?

or an inidvidual service?

How does service developer register and configure a service?

How does tenant admin signup and enable services?

How does user visit tenant specific service from the portal?

What are available responsibilities?

What responsibilities lie with the portal vs. the service developer?

How is self-service enabled for service developer and tenant admin?

Emphasis on...

Security, loose coupling

Separation of portal from service.

Bring your own app vs. fit to platform (PaaS).

High level usage attributes.

Particular emphasis is on security and loose coupling of services and their components. For example, in a VoIP system, if a SIP gateway is used, it should be possible to easily replace the gateway with that from another vendor, without breaking rest of the system.

Separting the portal from the service allows keeping the portal simple, and still be able to handle a wide range of services. In particular, we want to allow bring-your-own-app or -service model, where teams within Avaya can create a service without cloud consideration, and make it cloud ready easily using the ALICE guidelines. Some specific set of guidelines are listed in the paper.

This is unlike a PaaS model, where the service must be modified to fit a particular platform. In our case such modification is expensive, particularly when the service developer is not sure whether to productize the app or not, before an initial customer trial.

So what is different in this from what's out there?

What's different?

SaaS, but not just one service

Services are independent

Focus on connection driven services

Delegate to service developer

Billing to tenant admin
not service developer or end user

First, ALICE is like software-as-a-service model, but instead of just one service, it is an architecture to allow many experimental services for trial. The architecture is suitable for organizations that churn out many apps or services and have many small or medium businesses as customers. The services are generally trusted by the portal.

These services are assumed to be independent, hence integrated service provisioning and monitoring are not done. This works for existing small and medium businesses requirements.

Our focus is on connection driven services, particular for self-service multi-tenancy. The provisioning and monitoring requirements and constraints are different than existing data driven cloud services.

A number of responsibilities are delegated from the portal to the service developer. This keeps the portal manageable, while moving some complexity to the service developer. Thus, the service developer can best decide how to handle those, e.g., for multi-tenancy, the provisioning API could be mapped to a multi-instance one within a specific service.

Unlike existing cloud model, where the service developer or end user are usually billed for the service, our system attempts to bill the tenant admin. This is particularly important for our set up where individual developer teams do not want to handle billing, and tenant users and service developers are decoupled from the billing aspect.

I have talked about the six specific services currently hosted in ALICE. Each of these services in turn include multiple servers. This shows the hosted architecture of the current implementations.

It uses Amazon virtual private cloud for some crucial services such as authentication or VoIP trunking. Some services may interconnect with third-party services, e.g., for storage or messaging, or even an on-premise VoIP gateway of the customer, in a hybrid cloud model.

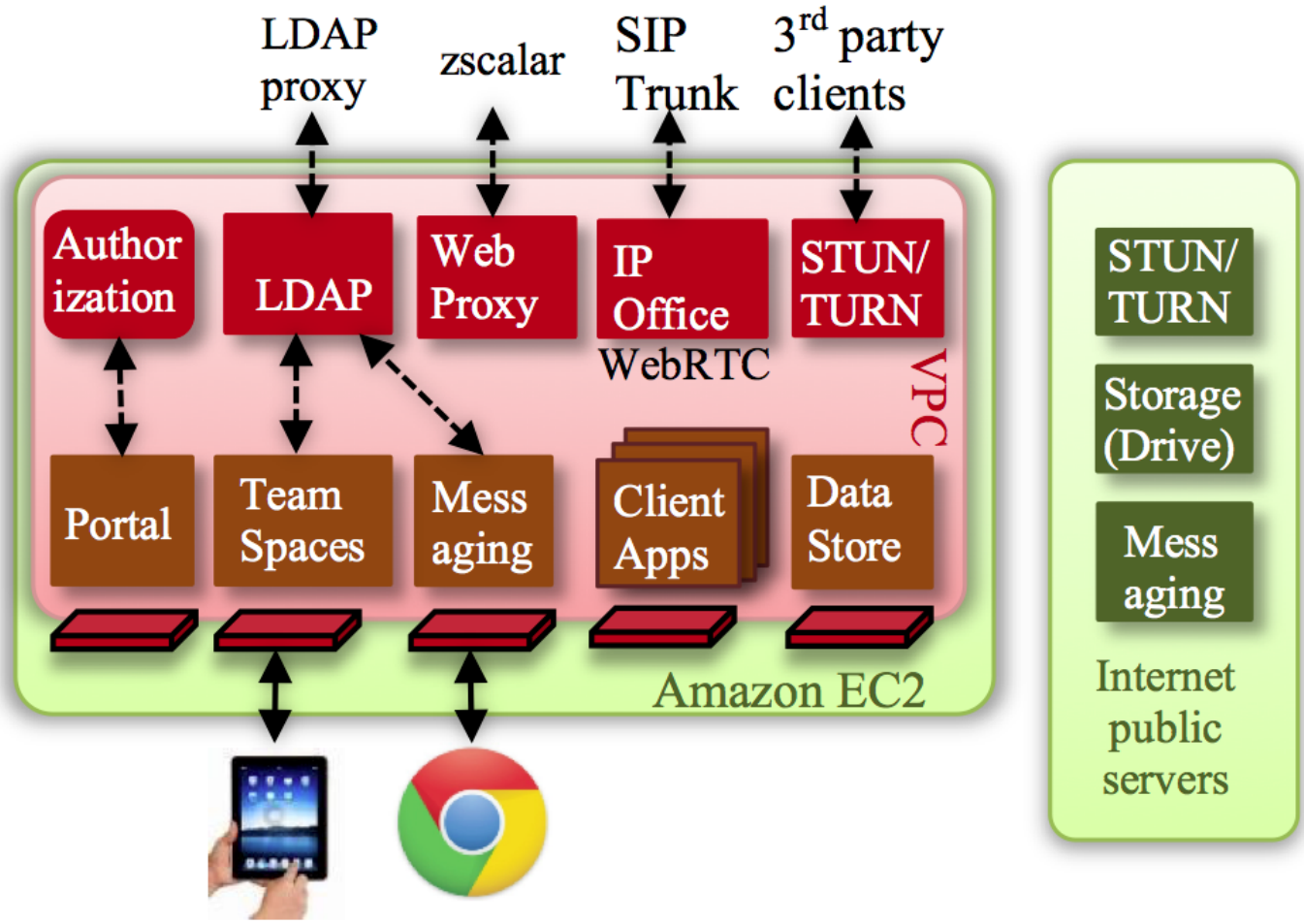


Figure 2. Current implementation contains several cloud services accessible from web or mobile.

Notes on services

1. Independent servers and DB,
   repeat for cross-app
2. Distributed,
   cloud, on-premise, hybrid
3. multi-tenant vs -instance
4. Browser, mobile app,
   thin-client vs. rich endpoint

5. Auth login vs. token, no SSO

As mentioned before, we try to duplicate the servers if there are overlaps, e.g., a resource server used in both Vclick and team spaces, and is duplicated for the two services.

Although the portal is multi-tenant, individual service may not be. It may be multi-instance or many require manual multi-tenant provisioning.

Most of our services are accessed using browsers and/or mobile clients. They include both thin-client and rich endpoints.

Finally, we don't force a particular authentication model. In fact one service does not even need any authentication. Some use LDAP style login, and others user authentication token. The authentication token allows multiple tokens per user login, and hence allows customizing separate devices of the same user.

The paper contains several tips and tricks to handle non-trivial challenges in enterprise cloud systems, especially related to compliance (or security), complexity of cloud and real-time aspects, cost, and compatibility with IT infrastructure.

Bag of tricks...

Challenges in compliance, complexity, cost, IT compatibility

Unsupported client certificate in a service

Incorrect cross-origin handling in a service

Load insecure http document in secure https service

Avoid password leak by accidental reuse on other sites

Reach private LDAP from a public service

Rogue ssh attempts fill up log files

Incorrect IP address due to NATed server

Local development and testing on single machine

We require client certificate to access some of the sensitive services for added access control. If the service does not support client certificate, how do we enable the feature in the portal, without modifying the service software?

Similarly, if cross-origin access is desired but not implemented or incorrectly implemented in the service, how do we enable it?

The team spaces service allows sharing documents among team members. These documents could be on insecure http web URLs to be shown inside a secure https service website. Since browsers do not allow such embedding, how do we enable sharing http content on an https service.

Using the same password in different services has the danger of leaking it. How can a service protect its users from such accidental leakage?

Some services hosted on public Internet may need to access private network, e.g., to authenticate using enterprise LDAP. How do we support such systems in a genaric manner independent of a specific protocol like LDAP?

Amazon cloud instances are common targets of attackers, especially looking to gain root access via ssh. Enabling public key based login instead of password, mitigates the attack. But this still fills up the crucial log files with garbage ssh attempts. How do we limit such ssh attempts to protect the log files?

Running a server behind a NAT, as in Amazon EC2 virtual private cloud, has the risk of exposing unreachable private IP address in application protocol, e.g., SDP. What techniques are available, what works and what does not?

Finally, due to the complexity of cloud and real-time development, local development and testing environment is highly desired. What challenges faced and solutions used to create and test such systems on a single machine?

These and other questions are answered in the paper. So I encourage you to read it, if the problems we faced are interesting to you too.

This brings me to the end of my presentation.

Questions?
Feedback?