AWS Visualize

Goal: Deliver a secure, simple, visually striking, useful, informative, fast, and intuitive alternative user interface for AWS.

Abstract: The AWS console is an extremely data rich UI, which has enabled thousands of users to the easier use of AWS products. It provides a table structured, product-centric, “spreadsheet” view of the resources in use and features available, and follows predominantly “wizard” style interactions for users to provision new resources.

When using a small (typically single-digit) number of resources, in a single region, and these views are fairly approachable; similarly, if you’re familiar with what the resources are being used for, it’s relatively straight-forward to track what’s going on inside an AWS account. Remove those two assumptions, and the console quickly becomes a very complex, difficult to digest, unintuitive view for many of our largest users.

We propose that a more visual user-interface mode, which emulates our Architecture Diagram style, and allows multi-product views of full customer systems running in AWS, would radically improve the user experience under these conditions.

It’s our intention to build this user interface as a stand-alone web application, which accepts IAM credentials and interrogates the AWS “describe” API’s, service health dashboard, and Cloudwatch to render isometric architecture views dynamically from the actual resources running in a customer’s account. It is also our goal to be able to visualize resources described via Cloudformation in the same view style.

Desired outcomes:

1. Customers will be able to see their infrastructure.
   1. Customers can see “at a glance” if any specific groups of their systems are experiencing issues (by region, role, owner, instance type, AMI, etc)
   2. Unified regional and stack level “navigation” to ensure that you’re seeing together the systems that are working together, no matter what products make up that system.
   3. Users unfamiliar with the structure of a given set of systems should be able to much more rapidly deduce “how things are laid out”
   4. Users will discover provisioning anomalies, misconfigurations and unintentional usage more easily because they can see things “out of place”
   5. With Cloudformation support, customers will have a method to validate what they think a script is going to do before they run it
   6. Help customers understand their footprint.
2. Loose the tables, keep the wizard
   1. We want to drive to existing console provisioning experiences, they’re fast and easy, but we want to trigger those from a visual UI: Drag and Drop architecture icons in place and drive wizard provisioning events that capture the customer’s directions from the visual UI.
   2. Right-click dialogs from the existing consoles are all good, action oriented behaviors, which we want to deep link to as well.
3. Metrics are embedded in the visuals
   1. KPI values overlaid and auto-updating
   2. Spark-graph versions of critical metrics superimposed on the infra-components
   3. Cloudwatch Alarms force color shifts on arch components, service health dashboard changes drive “area” color shifts.

Ideas

1. Regions in use are shown, those that aren’t, are hidden (only visible when provisioning new resources). No pull down menus; each running region of infra is a cluster of icons that can be zoomed into.
2. Groupings revolve around
   1. ELB registration
   2. route 53 record set target(s)
   3. Auto-Scaling Groups
   4. Security Groups
   5. Subnets
   6. Tags
   7. AMI
3. Connections between groups are created by
   1. Security Groups
   2. Route Tables
   3. VGW/IGW
   4. EIP
   5. ELB registration
4. Connections have a direction of traffic (or are bi-directional), that should be depicted!
5. Icons of instances are sized by instance type; high-mem are deep boxes, high-cpu are tall, high-net are wide (cc); same with RDS, elasticache, EMR..
6. A really critical concept is the idea of “steps” in an infrastructure (first an incoming user hits our DNS, then ELB, then our security group, then the web-tier…)we should drive customers to tag stacks with a number so we (and others!) know which order *they think* traffic is going in, and we can validate that against their security groups…
7. Drill-up and drill-down needs to be a very simple double-click or mouse-wheel action, from system to region, from region to groups, from front groups to back groups or from group to instance/service. Metrics need to be available at each of these tiers in appropriate aggregations. Outliers should be highlighted! A “side-step” navigation to the salient AWS console should be available at each level as well.
8. Aside from the zoom behaviors, customers still need the ability to filter using search phrase against tags/amis/ID#’s/security groups, which should be easily toggle-able on and off (hidden/viewed) and find based on the same search phrase (water ripple around matching resources)
9. Layout system really needs to support very high screen resolutions (retina, perceptive pixel, etc) for NOC-style usage, but still make sense to users at 1280x800
10. Whole services should also be able to be hidden/viewed
11. “viewport” information should be a savable state; shareable by querystring on a URL. Customers should be bookmarking views.
12. Metrics can be sparklines on the icons, bar charts off the edge of the icons, KPI’s on the faces of them, or hovering upper right like badge notifications.

Speed of this user interface is extremely critical. Where at all possible the UI should be optimized for prompt visual reaction to user actions, and clearly highlight where data is still loading. Data from the API’s should be cached in near-presentation format in DynamoDB/elasticache for low-latency access from the UI.

While quite a few structures can be discerned by the combination of AMI meta-data, (pretty easy to guess that the instances running the SQL Server AMI, having the SQL Server ports open in their security group should end up in the back row of the n-tier arch diagram) it is our expectation to drive adoption of Tags as a system for adding important categorization and role information to AWS resources within Visualize.

AWS, and specifically SA Carlos Conde has built a layout style around a simple iconography which uses an isometric view to increase density in architecture diagrams. We would use this design language as the core component of our views, while lifting interface styling queues from the existing console.