Digital Services Playbook Response

Play: 1

Understand what people need

We must begin digital projects by exploring and pinpointing the needs of the people who will use the service, and the ways the service will fit into their lives. Whether the users are members of the public or government employees, policy makers must include real people in their design process from the beginning. The needs of people — not constraints of government structures or silos — should inform technical and design decisions. We need to continually test the products we build with real people to keep us honest about what is important.

Checklist

1. Early in the project, spend time with current and prospective users of the service
2. Use a range of qualitative and quantitative research methods to determine people's goals, needs, and behaviors; be thoughtful about the time spent
3. Test prototypes of solutions with real people, in the field if possible
4. Document the findings about user goals, needs, behaviors, and preferences
5. Share findings with the team and agency leadership
6. Create a prioritized list of tasks the user is trying to accomplish, also known as "user stories"
7. As the digital service is being built, regularly test it with potential users to ensure it meets people's needs

SymSoft Solutions Response

1. At the outset of the project we recruited and performed user interviews with several citizens that would be likely users of the Cal-Notify alert system. Subsequently, these individuals participated in storyboarding and iterative testing as our prototype was designed and developed. Their insights were the key driver behind our resulting design.
2. We conducted interviews to understand user points of view, obtain insights into end user wants and needs and to provide valuable feedback on both the user interface design as well as the feature set of the prototype. We also conducted an

online survey to provide foundational quantitative information for the prototype design. The survey showed which alerts (e.g. fire, flooding, earthquakes) were the most concerning to citizens and the communication methods most relevant to them, segmenting responses by those who had or had not had first hand experiences with these events. These survey results provided valuable insight into priorities and preferences from real world users that helped us prioritize.

3. We had several users involved in user story development and usability testing of our prototype throughout the iterative design and development process.
4. We documented results of our user-centered design activities on GitHub
5. Results from user interactions were consistently shared including during our sprint planning meetings, daily standups and sprint reviews. User input was integrated into our development sprints as checkpoints against completed functionality during sprint reviews (e.g. we measured whether or not the end product captured the insights gained from our user research).
6. We created user stories and managed the resulting product development backlog using Jira and documented user-centered design activities in GitHub.
7. User interactions informed every aspect of our product development from early concept sketches and storyboards through iterative development and usability testing of our finished prototype. This includes both the public-facing alert system as well as the back-end administrative interface.

Key questions

1. Who are your primary users?
2. What user needs will this service address?
3. Why does the user want or need this service?
4. Which people will have the most difficulty with the service?
5. Which research methods were used?
6. What were the key findings?
7. How were the findings documented? Where can future team members access the documentation?
8. How often are you testing with real people?

SymSoft Solutions Response

1. Our users for Prototype B could be any citizen impacted or otherwise interested in staying informed about natural disasters so we had a very broad pool to draw from. We received input from dozens of users, inclusive of our quantitative user

survey and detailed insights from a pool of users actively involved in our user-centered development process onsite in our Sacramento office.

2. Our research showed helping people receive real-time information about natural disasters such as flooding, earthquakes and fires can help them plan travel routes, avoid being impacted by potentially harmful events and receive real-time information about events that may impact their loved ones. There were also indications that such a service could help relief efforts by helping citizens to avoid areas where emergency services and relief work may be underway. Our research indicated that a Cal-Notify service would provide a tremendous bi-directional exchange of value between Government and the citizens of California.
3. Our research and user centered design process showed that citizens want information about natural disasters for a number of reasons, including travel planning, understanding circumstances surrounding loved ones and plain curiosity. Our survey showed that nearly everyone who knew of the Cal-Notify system would choose to opt-in to receive alerts via text messaging. This includes those who had and those who had not ever been directly impacted by a natural disaster such as a flood, fire or earthquake.
4. Since by far the most desirable method of receiving alerts was through text messaging, those without access to a mobile device that included the ability to text would presumably be the most underserved by the Cal-Notify system.
5. Our user research methods included in-depth user interviews, persona development, collaborative storyboarding, and iterative test-driven development with real users, a quantitative survey and usability testing of the finished prototype.
6. Key findings were that there was nearly universal interest in receiving emergency alert notifications among broad user segments (e.g. those who had and those who had not been directly impacted by a natural disaster). The most desired means of communication by far was text messaging. Another key finding is that people were highly interested in opting in to the service, they were also adamant about the ability to opt-out. Generally speaking, users were not concerned about "Help" responses pertaining the application itself in an SMS opt-in strategy.
7. Our work is documented on GitHub including artifacts from our iterative, user-centered development approach.
8. We tested with real-world users at the end of each development sprint and conducted usability testing on the completed prototype including public facing and back-end administrative components.

Play 2

Address the whole experience, from start to finish

We need to understand the different ways people will interact with our services, including the SymSoft Solutions Response they take online, through a mobile application, on a phone, or in person. Every encounter — whether it's online or offline — should move the user closer towards their goal.

Checklist

1. Understand the different points at which people will interact with the service – both online and in person
2. Identify pain points in the current way users interact with the service, and prioritize these according to user needs
3. Design the digital parts of the service so that they are integrated with the offline touch points people use to interact with the service
4. Develop metrics that will measure how well the service is meeting user needs at each step of the service

SymSoft Solutions Response

1. We conducted initial user interviews and a survey to gain initial understandings about how users in the real world would foresee interacting with the State of California during a natural disaster. We learned right away that most users expected some form of interaction via their mobile device and that SMS was the preferred method for the vast majority of users. We learned that the desire to receive notifications was not driven by first-hand experiences with natural disasters in that a majority of users including those with no first-hand experience with natural disasters would nonetheless opt-in to Cal-Notify. Text was by far the most desirable communication method although there were those who preferred email and those who preferred not to receive notifications at all, but rather, access the information on their own timeframe from a website.
2. As a result of user interviews, we developed high level personas and used those personas in the creation of storyboards, use cases and scenarios. These activities helped us to isolate user needs as well as understand potential friction points in the adoption of an alert system (e.g. users who do not like to opt-in to text messaging out of privacy concerns). These insights influenced our prototype

design but also demonstrated the need for a user-centered design process with real users if and when The State should decide to actually implement a Cal-Notify system.

3. None of the users we interviewed proactively sought offline sources of information about natural disasters. They became aware from news reporting and social networks. They typically tune in to news sources after hearing about an event from their social network. Almost all respondents to our survey were interested in receiving alerts from The State regarding natural disasters.
4. We integrated Google Analytics on the Cal-Notify alert signup page in order to track and monitor activity on the site. We would also recommend for an actual implementation that techniques such as A/B testing should be used from the start of the rollout to allow The State to optimize enrollment as well as using techniques such as feedback surveys to understand any "friction" relative to adoption and retention of customers for the Cal-Notify system.

Key questions

1. What are the different ways (both online and offline) that people currently accomplish the task the digital service is designed to help with?
2. Where are user pain points in the current way people accomplish the task?
3. Where does this specific project fit into the larger way people currently obtain the service being offered?
4. What metrics will best indicate how well the service is working for its users?

Answers to key questions

1. We found in the users we interviewed that people typically learn about natural disasters from the news media and through their social networks (social media sites or word-of-mouth).
2. The largest pain point was simply the lack of a system to receive notifications and alerts and the majority of our survey respondents expressed interest irrespective of their own personal history with such events.
3. Cal-Notify could serve to inform citizens much earlier through official channels about natural disasters as opposed to learning about them through delayed sources such as the news media and social networks. Having specific information relative to geography was important to users, especially as it relates to pending travel plans and the disposition of loved ones in the impacted area. Precise geographic information is not available from their current sources.

4. Metrics to indicate how well the service is working would be customer experience oriented metrics for enrollments such as the number of visits to the enrollment site versus the number of actual subscribers and the number of total potential subscribers versus the number of actual subscribers. Other metrics such as the rate of opt-outs would be useful in identifying potential issues with the customer experience. The number of “Help” requests would also be important but this type of metric needs to be used in a “closed loop” where the user requesting help is tracked and followed up with. Unresolved issues is a primary reason for online subscription service drop offs in general and it is extremely difficult to reenroll a customer once they’ve elected to drop off of a subscription.

Play 3

Make it simple and intuitive

Using a government service shouldn't be stressful, confusing, or daunting. It's our job to build services that are simple and intuitive enough that users succeed the first time, unaided.

Checklist

1. Create or use an existing, simple, and flexible design style guide for the service
2. Use the design style guide consistently for related digital services
3. Give users clear information about where they are in each step of the process
4. Follow accessibility best practices to ensure all people can use the service
5. Provide users with a way to exit and return later to complete the process
6. Use language that is familiar to the user and easy to understand
7. Use language and design consistently throughout the service, including online and offline touch points

SymSoft Solutions Response

1. We used Bootstrap as the foundation for developing our Cal-Notify user interface. Due to its simplicity and open source availability, Bootstrap has become one of the most, if not the most, widely used framework for developing responsive websites and applications in the industry. Bootstrap is open source and is developed, and maintained on GitHub so it is freely available to the State and well-aligned with the State's requirement to use open source technology. Our application of Bootstrap to the Cal-Notify user interface is documented in our Style Guide.
2. Bootstrap and our associated style guide information was used consistently for the Cal-Notify user interface.
3. Cal-Notify enables users to sign up for email or SMS alerts based on their chosen location as well as to maintain their profile and discontinue receiving alerts. The profile screen shows which alert type they are enrolled in (email or SMS) and provides a way to opt-out. Opt-out by text "STOP" was not included in the prototype but should be in the actual system. Opt-in by Short Code should also be considered which could provide The State another way to attract and enroll subscribers.

4. We used a mix of both automated and manual testing by internal staff who are subject matter experts on Section 508 and W3C accessibility guidelines to help ensure that the Cal-Notify application complies with Section 508 and WCAG 2.0 AA priority.
5. In the case of Cal-Notify, the enrollment process is a single step process.
6. We tested our Cal-Notify product with real users who confirmed their ease of understanding and we utilized an internal copyediting subject matter expert to develop content that was influenced based on user feedback.
7. Cal-Notify is designed to be an online service. There are no offline components. As mentioned above, Opt-in by Short Code could be useful in support an offline method to attract subscribers (e.g. could be on a poster at DMV offices, for example).

Key questions

- What primary tasks are the user trying to accomplish?
- Is the language as plain and universal as possible?
- What languages is your service offered in?
- If a user needs help while using the service, how do they go about getting it?
- How does the service's design visually relate to other government services?

Answers to key questions

1. In Cal-Notify, users are trying to subscribe to the service in order to receive alerts. Users are prompted via SMS to confirm their enrollment with a link to a simple screen where they can set their password and validate maintain their preferences.
2. We enlisted internal copyediting specialist to ensure the language is as plain and universal as possible and usability tested the language with real users.
3. Cal-Notify prototype currently supports English. It would be appropriate and not greatly difficult to include multi-language support for the actual application.
4. While the Cal-Notify sign-up process is very simple, usability testing identified that the address field requested upon enrollment required some explanation. More specifically, when users provide a specific street address it is used to determine when they receive alerts (this could be their home address or any other address that they are concerned about, such as the home of a loved one). To address this usability testing "find", we included online help that explains the field and how the information is used. We included privacy information per opt-in best practices. For the administrative interface, we used inline explanatory text.

5. For the Cal-Notify prototype we did not attempt to align visual design with other government services. Bootstrap styling could be readily enhanced to align with other government services or style guidelines.

Play 4

Build the service using agile and iterative practices

We should use an incremental, fast-paced style of software development to reduce the risk of failure. We want to get working software into users' hands as early as possible to give the design and development team opportunities to adjust based on user feedback about the service. A critical capability is being able to automatically test and deploy the service so that new features can be added often and be put into production easily.

Checklist

1. Ship a functioning "minimum viable product" (MVP) that solves a core user need as soon as possible, no longer than three months from the beginning of the project, using a "beta" or "test" period if needed
2. Run usability tests frequently to see how well the service works and identify improvements that should be made
3. Ensure the individuals building the service communicate closely using techniques such as launch meetings, war rooms, daily standups, and team chat tools
4. Keep delivery teams small and focused; limit organizational layers that separate these teams from the business owners
5. Release features and improvements multiple times each month
6. Create a prioritized list of features and bugs, also known as the "feature backlog" and "bug backlog"
7. Use a source code version control system
8. Give the entire project team access to the issue tracker and version control system
9. Use code reviews to ensure quality

SymSoft Solutions Response

1. By the end of our First sprint we released an initial version of the prototype which solved our user's core needs by allowing them to receive notifications. The first sprint provided an ability for Administrative user to manually create a notification using a simple user interface. By the end of our second sprint we integrated all the feedback gathered during the testing of our initial version, and added feature to pull the data from USGS (earthquakes) and NOAA (river floods). We put this version of our prototype in user's hands within two weeks.

2. We ran multiple usability tests during the development of our prototype starting with the interactive Balsamiq wireframes, an initial HTML prototype and the consecutive iterations of the actual prototype. On each test, we identified areas of improvement and integrated those improvements on our next iteration. Based on the usability tests, we modified the registration screen to provide in-context information about how the user-provided data will be used, which is something the users were interested in knowing.
3. We kept constant and clear communication not only across the team but also with the prototype users. We performed all the activities of the Scrum framework such as Daily Stand-ups, Sprint Planning, Sprint Retrospectives etc. We also enabled a Slack channel and integrated it with our issue tracker (Jira), continuous integration and continuous integration system (Jenkins CI) so that the team was informed of the progress of the project and the valued provided to the users
4. Our delivery team has=d no management layers and some members of the team played multiple roles in this effort. The team included front-end and Back-end developers, as well as a content strategist, product manager, interaction designer / developer, visual designer, delivery manager security specialist and others.
5. Our continuous delivery strategy allowed us to deploy working code to the live instance multiple times a day as demonstrated on GitHub. The features were planned for each sprint and released with completion of the sprint.
6. Our Scrum Master and Product Manager kept the backlog prioritized and up-to-date at all times. We used Jira to manage this.
7. We used GitHub as our source code version control system. Each of our team members have access to the project repository.
8. The entire project team had access to both Jira and GitHub. We used Jira with the Scrum Software Development Project type for documenting all our epics, user stories, tasks and bugs. We used GitHub for source code version control.
9. Code reviews were continuously performed by peers of team members and by our Technical Architect with every user story that was completed.

Key questions

1. How long did it take to ship the MVP? If it hasn't shipped yet, when will it?
2. How long does it take for a production deployment?
3. How many days or weeks are in each iteration/sprint?
4. Which version control system is being used?
5. How are bugs tracked and tickets issued? What tool is used?
6. How is the feature backlog managed? What tool is used?

7. How often do you review and reprioritize the feature and bug backlog?
8. How do you collect user feedback during development? How is that feedback used to improve the service?
9. At each stage of usability testing, which gaps were identified in addressing user needs?

Answers to key questions

1. We put our MVP in users hands by the end of Sprint 1.
2. For a production deployment of the prototype, we only need to trigger the deployment from our continuous integration server (Jenkins) by clicking a button. It will execute all tests and deploy the code to the production environment in less than 5 min. No manual steps are required.
3. We ran a total of 4 sprints that were each 5 days. Sprint durations typically range between 2 and 4 weeks depending on the project but for this prototype, the sprint duration we agreed upon was 1 week which was helpful in addressing the compressed timeframe for development mandated by the RFI while also have enough iterations of user feedback during development.
4. We used Git.
5. All bugs/issues are tracked using Jira. A reported issue provides information such as a detailed description of the issue, steps to reproduce, testing environment details, screenshots etc.
6. The Scrum Master and the Product Manager manage the backblock in Jira. They make sure every item in the backlog is sized, prioritized and assigned accordingly.
7. Everyday for this prototype. The frequency may vary based on sprint duration, which in turn depends on the complexity of the project/user stories. Typical sprint durations are between 2-4 weeks, though for this prototype, the sprint duration was 1 week as agreed to by the team prior to development starting.
8. We collected feedback from user testing different iterations of our application and documented that feedback in video and audio recordings of usability testing. The collected feedback is analyzed and converted into user stories, bugs, tasks that are prioritized and executed as agreed to by the team and Product Manager. The result of this effort is also presented to users for validating that the changes address the initial user issues.
9. We identified multiple needs which we addressed, most of the issues were related to simplifying the use of the back-end administrative interface and the way that alerts information is communicated.

Play 5

Structure budgets and contracts to support delivery

To improve our chances of success when contracting out development work, we need to work with experienced budgeting and contracting officers. In cases where we use third parties to help build a service, a well-defined contract can facilitate good development practices like conducting a research and prototyping phase, refining product requirements as the service is built, evaluating open source alternatives, ensuring frequent delivery milestones, and allowing the flexibility to purchase cloud computing resources.

The TechFAR Handbook provides a detailed explanation of the flexibilities in the Federal Acquisition Regulation (FAR) that can help agencies implement this play.

Checklist

1. Budget includes research, discovery, and prototyping activities
2. Contract is structured to request frequent deliverables, not multi-month milestones
3. Contract is structured to hold vendors accountable to deliverables
4. Contract gives the government delivery team enough flexibility to adjust feature prioritization and delivery schedule as the project evolves
5. Contract ensures open source solutions are evaluated when technology choices are made
6. Contract specifies that software and data generated by third parties remains under our control, and can be reused and released to the public as appropriate and in accordance with the law
7. Contract allows us to use tools, services, and hosting from vendors with a variety of pricing models, including fixed fees and variable models like “pay-for-what-you-use” services
8. Contract specifies a warranty period where defects uncovered by the public are addressed by the vendor at no additional cost to the government
9. Contract includes a transition of services period and transition-out plan

SymSoft Solutions Response

1. Our assigned product manager created and managed the budget for the development of this prototype.
2. NOT APPLICABLE TO THIS PROTOTYPE
3. NOT APPLICABLE TO THIS PROTOTYPE
4. NOT APPLICABLE TO THIS PROTOTYPE
5. NOT APPLICABLE TO THIS PROTOTYPE
6. NOT APPLICABLE TO THIS PROTOTYPE
7. NOT APPLICABLE TO THIS PROTOTYPE
8. NOT APPLICABLE TO THIS PROTOTYPE
9. NOT APPLICABLE TO THIS PROTOTYPE

Key questions

1. What is the scope of the project? What are the key deliverables?
2. What are the milestones? How frequent are they?
3. What are the performance metrics defined in the contract (e.g., response time, system uptime, time period to address priority issues)?

Answers to key questions

1. The scope of the project can be summarized as a public facing alert sign up, ability to deliver and receive SMS and email alerts and an administrative back-end system to operate the alert prototype. The scope of these functions was limited based on time allotted (please refer to the RFI for more specifics)
2. Our primary milestone was usability testing iterations of development with real users along with completing the backlog items that resulted from these tests. For an actual implementation, milestones would include post-implementation monitoring and optimization of actual enrollments in the service.
3. NOT APPLICABLE TO THIS PROTOTYPE BEYOND RFI REQUIREMENTS

Play 6

Assign one leader and hold that person accountable

There must be a single product owner who has the authority and responsibility to assign tasks and work elements; make business, product, and technical decisions; and be accountable for the success or failure of the overall service. This product owner is ultimately responsible for how well the service meets needs of its users, which is how a service should be evaluated. The product owner is responsible for ensuring that features are built and managing the feature and bug backlogs.

Checklist

1. A product owner was assigned.
2. All stakeholders agree that the product owner has the authority to assign tasks and make decisions about features and technical implementation details.
3. The product owner has a product management background with technical experience to assess alternatives and weigh tradeoffs.
4. The product owner has a work plan that includes budget estimates and identifies funding sources
5. The product owner has a strong relationship with the contracting officer

SymSoft Solutions Response

1. We assigned Savita Farooqui as the single product owner.
2. All stakeholders agreed that he product had the authority to assign tasks and make decisions about features and technical implementation details.
3. Our product owner has 20+ years of experience in product development with an emphasis on Government products and has played a lead technical architect and hands-on development role in many projects over the years.
4. The product owner worked with the team and established the delivery plan and funding sources for the multi-disciplinary team we used to create the prototype.
5. Not applicable to the prototype phase

Key questions

1. Who is the product owner?
2. What organizational changes have been made to ensure the product owner has sufficient authority over and support for the project?
3. What does it take for the product owner to add or remove a feature from the service?

Answers to key questions

1. We assigned Savita Farooqui as the single product owner.
2. Our product owner being the CEO, we did not require organizational changes in order to ensure the product owner has sufficient authority over and support for the project.
3. The team agreed the product owner was granted the ultimate decision-making authority over prototype features. Since we used user-centered design, decisions were driven by our design process with users including the team input based on user interactions as opposed to unilateral decisions by the product owner

Play 7

Bring in experienced teams

We need talented people working in government who have experience creating modern digital services. This includes bringing in seasoned product managers, engineers, and designers. When outside help is needed, our teams should work with contracting officers who understand how to evaluate third-party technical competency so our teams can be paired with contractors who are good at both building and delivering effective digital services. The makeup and experience requirements of the team will vary depending on the scope of the project.

Checklist

1. Member(s) of the team have experience building popular, high-traffic digital services
2. Member(s) of the team have experience designing mobile and web applications
3. Member(s) of the team have experience using automated testing frameworks
4. Member(s) of the team have experience with modern development and operations (DevOps) techniques like continuous integration and continuous deployment
5. Member(s) of the team have experience securing digital services
6. A Federal contracting officer is on the internal team if a third party will be used for development work
7. A Federal budget officer is on the internal team or is a partner
8. The appropriate privacy, civil liberties, and/or legal advisor for the department or agency is a partner

SymSoft Solutions Response

1. For this prototype, SymSoft assembled an experienced and skilled team. The team comprised of key members of the team have experience building high-traffic web portals and digital services including the California's state portal Ca.gov that provides access to all state and local services, and serves 37 million Californians, with over 10 million page views per month.
2. Our team members have experience building mobile applications for both iOS and Android, as well as responsive web applications that work well on mobile devices.

3. The team routinely follows test-driven development and uses automated testing frameworks (it is a required process for all our projects). For this prototype, we used xUnit for automated unit tests.
4. The team has deep experience with DevOps having used continuous integration and deployment scenarios in all our projects. This is the only way deployments happen. For this project, we utilized Jenkins.
5. The team includes the company's Security Officer, as well as other members of the security team who have experience with and are responsible for securing digital services.
6. NOT APPLICABLE FOR THIS PROTOTYPE.
7. NOT APPLICABLE FOR THIS PROTOTYPE.
8. NOT APPLICABLE FOR THIS PROTOTYPE.

Play 8

Choose a modern technology stack

The technology decisions we make need to enable development teams to work efficiently and enable services to scale easily and cost-effectively. Our choices for hosting infrastructure, databases, software frameworks, programming languages and the rest of the technology stack should seek to avoid vendor lock-in and match what successful modern consumer and enterprise software companies would choose today. In particular, digital services teams should consider using open source, cloud-based, and commodity solutions across the technology stack, because of their widespread adoption and support by successful consumer and enterprise technology companies in the private sector.

Checklist

1. Choose software frameworks that are commonly used by private-sector companies creating similar services
2. Whenever possible, ensure that software can be deployed on a variety of commodity hardware types
3. Ensure that each project has clear, understandable instructions for setting up a local development environment, and that team members can be quickly added or removed from projects
4. Consider open source software solutions at every layer of the stack

SymSoft Solutions Response

1. We used Microsoft .NET Core for the foundation of our application. Data is stored in a Postgres/PostGIS database to support geofence calculations. Swagger is used in the .NET core API layer. For the application's front-end (both for the public and for administrative back-end), we utilized Bootstrap and AngularJS. All of these technologies are commonly used by private-sector companies. We ourselves have used these technologies for our private sector client projects.
2. The technology stack used for this prototype is compatible with Linux and Microsoft server operating systems, as well as Docker. This ensures that it can be deployed on a variety of low cost, commodity hardware. This also ensures the ability to scale for performance as needed.

3. We have created simple project setup guidelines that help developers to quickly and easily get a local development environment ready and begin contributing to the project. This can be found on GitHub here: https://github.com/SymSoftSolutions/cal-notify/blob/master/back-end/README.md.
4. The front-end of the solution utilizes Bootstrap and AngularJS. The API layer of the application utilizes .NET core and Swagger. The database layer consists of a Postgres/PostGIS database. All of these technologies are open source. In addition, because we carefully chose the technologies listed above, the solution can be deployed to Linux or Docker, both of which are also open source technologies.

Key questions

1. What is your development stack and why did you choose it?
2. Which databases are you using and why did you choose them?
3. How long does it take for a new team member to start developing?

Answers to key questions

1. We chose .NET Core, AngularJS, and Swagger for the core of the application. The primary reason behind this was that they are open source technologies and well-used and supported in the software industry. Other reasons include:
    - .NET Core is lightweight and performant.
    - .NET Core is supported on multiple platforms such as Windows, Linux, and MacOS, making it easy for developers of all stacks to contribute.
    - Angular provides simple yet effective ways to present various types of data on the website and is efficient from a development perspective. It also integrates well with the Swagger API endpoints.
    - Swagger provides full documentation for the API, making it easy to communicate intended functionality to the project team.
2. We used PostgreSQL/PostGIS to house the database. Aside from being open source, this was chosen for two main reasons:
    - PostgreSQL is a lightweight and performant database engine.
    - PostGIS allows for complex geospatial queries, which we use for functionality that provides alerts to registered users.
3. Including installation of pre-requisites, a developer should be able to begin developing and contributing to the project in less than half a workday (4 hours).

Play 9

Deploy in a flexible hosting environment

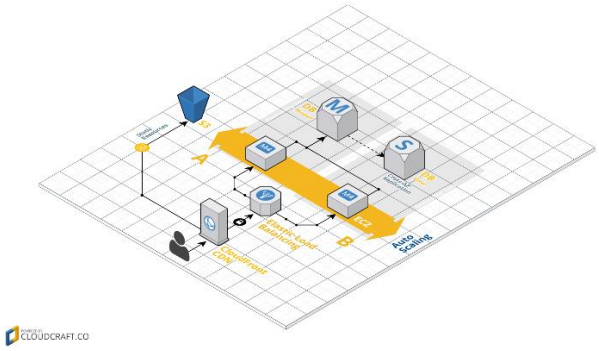
Our services should be deployed on flexible infrastructure, where resources can be provisioned in real-time to meet spikes traffic and user demand. Our digital services are crippled when we host them in data centers that market themselves as “cloud hosting” but require us to manage and maintain hardware directly. This outdated practice wastes time, weakens our disaster recovery plans, and results in significantly higher costs.

Checklist

1. Resources are provisioned on demand
2. Resources scale based on real-time user demand
3. Resources are provisioned through an API
4. Resources are available in multiple regions
5. We only pay for resources we use
6. Static assets are served through a content delivery network
7. Application is hosted on commodity hardware

SymSoft Solutions Response

The prototype B implementation is deployed in AWS in an EC2 instance. The instance for this prototype is a single instance. When the prototype is ready for deployment, following reference architecture will be used.



The proposed architecture scales based on real-time user demand and provides disaster recovery and business continuity through multiple availability zone deployments.

1. The current AWS instance is provisioned on-demand.
2. For this prototype auto-scaling of resources based on user-demand has not been implemented. We would deploy the production instance in the proposed architecture (refer above) that includes this provision.
3. Using AWS APIs allow automatic provisioning.
4. AWS regions have multiple availability zones. The availability zones are geographically dispersed. Most implementations use multi-availability-zone deployments of databases and application server instances to provide Disaster Recovery and Business Continuity. For this prototype a single self-contained server (UI, API and DB all in one server) has been deployed.
5. AWS charges by the minute.
6. AWS CloudFront provides a content delivery network for most of the countries, around the world.
7. AWS uses commodity hardware.

Key questions

1. Where is your service hosted?
2. What hardware does your service use to run?
3. What is the demand or usage pattern for your service?
4. What happens to your service when it experiences a surge in traffic or load?
5. How much capacity is available in your hosting environment?
6. How long does it take you to provision a new resource, like an application server?
7. How have you designed your service to scale based on demand?
8. How are you paying for your hosting infrastructure (e.g., by the minute, hourly, daily, monthly, fixed)?
9. Is your service hosted in multiple regions, availability zones, or data centers?
10. In the event of a catastrophic disaster to a datacenter, how long will it take to have the service operational?
11. What would be the impact of a prolonged downtime window?
12. What data redundancy do you have built into the system, and what would be the impact of a catastrophic data loss?
13. How often do you need to contact a person from your hosting provider to get resources or to fix an issue?

Answers to key questions

1. Amazon Web Services, US West (Oregon) data center.
2. The prototype runs on a Cloud T2 Small instance. See Instance Types for more details.
3. The prototype is for demonstration only. Therefore, the number of users are very small. In most of our deployments, we provision resources based on estimated number of users, to begin with and scale/de-scale based on the demand (in most cases scaling based on real-time demand is implemented.)
4. The current prototype deployment is designed for a volume adequate for the proof-of-concept.
5. AWS provides ability to scale up or scale-out the infrastructure as the needs grow.
6. Provisioning the server instances through interactive dashboard takes about 20 minutes. This process can be made faster by using automated provisioning scripts.
7. As mentioned earlier, we would implement the proposed architecture for the production deployment of the system, that includes scaling on-demand.
8. Amazon pricing, which has some fixed and some usage rates.
9. The current prototype is hosted in single region.
10. The prototype system can be reproduced in less than 30 minutes in any region. Since this is a prototype, we are not backing up data. However, for production systems we have successfully deployed systems with 12 minutes RPO and as high as 5 9's availability.
11. The prototype is for demonstration only. However, in the assessment period a prolonged downtime will adversely impact our evaluation.
12. The prototype fetches the alerts from web based public sources and allows administrators to update/customize the alert and publish them. The impact of a catastrophic failure would be negligible. However, if this prototype is scaled for production, then we would add appropriate cross-availability zone replication to reduce impact of database failures.
13. We did not have a need to contact AWS support for this prototype. However, in our production systems, we have created four to six support tickets per year

Play 10

Automate testing and deployments

Today, developers write automated scripts that can verify thousands of scenarios in minutes and then deploy updated code into production environments multiple times a day. They use automated performance tests which simulate surges in traffic to identify performance bottlenecks. While manual tests and quality assurance are still necessary, automated tests provide consistent and reliable protection against unintentional regressions, and make it possible for developers to confidently release frequent updates to the service.

Checklist

1. Create automated tests that verify all user-facing functionality
2. Create unit and integration tests to verify modules and components
3. Run tests automatically as part of the build process
4. Perform deployments automatically with deployment scripts, continuous delivery services, or similar techniques
5. Conduct load and performance tests at regular intervals, including before public launch

SymSoft Solutions Response

1. We developed tests using Selenium WebDriver API
2. We developed unit test using xUnit.
3. Our tests suite is automatically executed by Jenkins with every build, the results of the tests are communicated to the team via the project channel on Slack.
4. Every commit triggers a deployment to our development environment after passing all the tests. For a production deployment, we sometimes stipulate that the deployment needs to be manually triggered by clicking a button in Jenkins, however besides clicking that button no additional manual steps are required for deploying the application to the production environment.
5. We executed load testing at the end of each sprint and use Amazon CloudWatch to continuously monitor performance

Key questions

1. What percentage of the code base is covered by automated tests?
2. How long does it take to build, test, and deploy a typical bug fix?
3. How long does it take to build, test, and deploy a new feature into production?
4. How frequently are builds created?
5. What test tools are used?
6. Which deployment automation or continuous integration tools are used?
7. What is the estimated maximum number of concurrent users who will want to use the system?
8. How many simultaneous users could the system handle, according to the most recent capacity test?
9. How does the service perform when you exceed the expected target usage volume? Does it degrade gracefully or catastrophically?
10. What is your scaling strategy when demand increases suddenly?

Answers to key questions

1. About 50%
2. It depends on the issue but for a typical bug fix in this project to be built, tested and deployed took an average of 30 minutes.
3. It again depends of the feature but in average including the conversations across the team it takes about 2 to 3 hours.
4. Deployment to a production environment is manually triggered in Jenkins, and it is executed multiple times per day. An average of 5 times per day.
5. We use Selenium WebDriver API and xUnit
6. We use Jenkins CI
7. Cal-Notify would have potentially millions of subscribers. The number of simultaneous users would vary by system component. Initial high traffic for enrollment would be foreseen and alerts traffic would be based on how many people were signed up in the impacted geography. Estimates for concurrent users during initial enrollment would be up to 5000 given the short duration of the enrollment session while simultaneous alerts could number in the hundreds of thousands if not millions (e.g. earthquake in a major metro area) so the system should be scaled accordingly.
8. We did not execute this test for the prototype.
9. We did not execute this test for the prototype.

10. We have designed a strategy that uses Amazon CloudWatch for monitoring application resources and scaling using an elastic load balancer. However for the purpose of this prototype we did not implement this strategy.

Play 11

Manage security and privacy through reusable processes

Our digital services have to protect sensitive information and keep systems secure. This is typically a process of continuous review and improvement which should be built into the development and maintenance of the service. At the start of designing a new service or feature, the team lead should engage the appropriate privacy, security, and legal officer(s) to discuss the type of information collected, how it should be secured, how long it is kept, and how it may be used and shared. The sustained engagement of a privacy specialist helps ensure that personal data is properly managed. In addition, a key process to building a secure service is comprehensively testing and certifying the components in each layer of the technology stack for security vulnerabilities, and then to re-use these same pre-certified components for multiple services.

The following checklist provides a starting point, but teams should work closely with their privacy specialist and security engineer to meet the needs of the specific service.

Checklist

1. Contact the appropriate privacy or legal officer of the department or agency to determine whether a System of Records Notice (SORN), Privacy Impact Assessment, or other review should be conducted
2. Determine, in consultation with a records officer, what data is collected and why, how it is used or shared, how it is stored and secured, and how long it is kept
3. Determine, in consultation with a privacy specialist, whether and how users are notified about how personal information is collected and used, including whether a privacy policy is needed and where it should appear, and how users will be notified in the event of a security breach
4. Consider whether the user should be able to access, delete, or remove their information from the service
5. “Pre-certify” the hosting infrastructure used for the project using FedRAMP
6. Use deployment scripts to ensure configuration of production environment remains consistent and controllable

SymSoft Solutions Response

1. Not applicable to this prototype

2. Not applicable to this prototype
3. Not applicable to this prototype
4. Not applicable to this prototype
5. AWS infrastructure is FedRAMP certified, including in the region we are using.
6. SymSoft Solutions Response uses Docker and other Infrastructure as Code tools to automate and control the creation, configuration and deployment of development and production environments.

Key questions

1. Does the service collect personal information from the user? How is the user notified of this collection?
2. Does it collect more information than necessary? Could the data be used in ways an average user wouldn't expect?
3. How does a user access, correct, delete, or remove personal information?
4. Will any of the personal information stored in the system be shared with other services, people, or partners?
5. How and how often is the service tested for security vulnerabilities?
6. How can someone from the public report a security issue?

Answers to key questions

1. The service requires the user to provide an email address or phone number for SMS alerts. The user signs up to receive notifications and provides personal information such as email address, phone number and an address. The registration form communicates to user how the information will be used.
2. We only collect information required to fulfill an alert. The data could not be used in ways other than expected barring some data security breach.
3. Users can opt-out of alerts. While not included in the prototype due to time constraints, this opt-out would result in purging user data in the actual system.
4. No, this is clearly stated on the sign up page
5. The service follows Open Web Application Security Project (OWASP) guidelines for development. Our prototype is not yet tested by any third party for security vulnerabilities.
6. This was not addressed for the prototype but could be part of the design for the actual implementation based on State requirements

Play 12

Use data to drive decisions

At every stage of a project, we should measure how well our service is working for our users. This includes measuring how well a system performs and how people are interacting with it in real-time. Our teams and agency leadership should carefully watch these metrics to find issues and identify which bug fixes and improvements should be prioritized. Along with monitoring tools, a feedback mechanism should be in place for people to report issues directly.

Checklist

1. Monitor system-level resource utilization in real time
2. Monitor system performance in real-time (e.g. response time, latency, throughput, and error rates)
3. Ensure monitoring can measure median, 95th percentile, and 98th percentile performance
4. Create automated alerts based on this monitoring
5. Track concurrent users in real-time, and monitor user behaviors in the aggregate to determine how well the service meets user needs
6. Publish metrics internally
7. Publish metrics externally
8. Use an experimentation tool that supports multivariate testing in production

SymSoft Solutions Response

1. We are using the AWS CloudWatch monitoring for this purpose and we have enabled it to monitor our servers.
2. CloudWatch Monitoring of CPU, Memory, Disk Use and status etc.
3. Yes, CloudWatch can support this
4. For real-time monitoring, we configure Alarms in AWS to trigger notifications on status checks so that action can be taken quickly.
5. We integrated Google Analytics for real-time measurement and analytics that will help determine if the service meets user needs.
6. Not implemented for this prototype due to time constraints
7. Not implemented for this prototype due to time constraints

8. Not implemented for this prototype due to time constraints

Key questions

1. What are the key metrics for the service?
2. How have these metrics performed over the life of the service?
3. Which system monitoring tools are in place?
4. What is the targeted average response time for your service? What percent of requests take more than 1 second, 2 seconds, 4 seconds, and 8 seconds?
5. What is the average response time and percentile breakdown (percent of requests taking more than 1s, 2s, 4s, and 8s) for the top 10 transactions?
6. What is the volume of each of your service's top 10 transactions? What is the percentage of transactions started vs. completed?
7. What is your service's monthly uptime target?
8. What is your service's monthly uptime percentage, including scheduled maintenance? Excluding scheduled maintenance?
9. How does your team receive automated alerts when incidents occur?
10. How does your team respond to incidents? What is your post-mortem process?
11. Which tools are in place to measure user behavior?
12. What tools or technologies are used for A/B testing?
13. How do you measure customer satisfaction?

Answers to key questions

1. The key metrics for this service include, but are not limited to, page views, sign ups (for alerts), emergency/non-emergency messages sent, and users affected by each alert.
2. Not applicable to this prototype due to time.
3. AWS Cloudwatch.
4. For most pages on the website, we expect a less than 2 second response time. Breakdown is as follows:
    - 1s: 75%
    - 2s: 90%
    - 4s: 95%
    - 5s: 99%
5. Not available for this prototype due to time.

6. Not available for this prototype due to time.
7. For an actual implementation we would recommend uptime at the high end of industry standards (four to five 9's) but this would be up to the State
8. Not applicable to this prototype due to time.
9. Alerts are received via e-mail. Our DevOps team is setup to receive emails on their mobile phone.
10. Incidents are tracked via a separate ticketing system and assigned to DevOps engineers for analysis. Ticket assignment may change during the lifecycle of the issue. Each identified issue goes through a postmortem process to determine root cause, and how that can be avoided in the future.
11. The prototype is using Google Analytics to track navigation behavior for the users. In the production environments, we use other tools to track and in some cases dynamically change the website content based on user behavior (e.g. personalization)
12. Due to the short duration of the prototype, A/B testing tools were not implemented. However, usability tests were conducted with users after each sprint to get feedback and improve the prototype.
13. Customer satisfaction was primarily gauged via user interviews and tests conducted after each sprint.

Play 13

Default to open

When we collaborate in the open and publish our data publicly, we can improve Government together. By building services more openly and publishing open data, we simplify the public's access to government services and information, allow the public to contribute easily, and enable reuse by entrepreneurs, nonprofits, other agencies, and the public.

Checklist

1. Offer users a mechanism to report bugs and issues, and be responsive to these reports
2. Provide datasets to the public, in their entirety, through bulk downloads and APIs (application programming interfaces)
3. Ensure that data from the service is explicitly in the public domain, and that rights are waived globally via an international public domain dedication, such as the "Creative Commons Zero" waiver
4. Catalog data in the agency's enterprise data inventory and add any public datasets to the agency's public data listing
5. Ensure that we maintain the rights to all data developed by third parties in a manner that is releasable and reusable at no cost to the public
6. Ensure that we maintain contractual rights to all custom software developed by third parties in a manner that is publishable and reusable at no cost
7. When appropriate, create an API for third parties and internal users to interact with the service directly
8. When appropriate, publish source code of projects or components online
9. When appropriate, share your development process and progress publicly

SymSoft Solutions Response

1. Our GitHub repository is open and allows external users to submit bugs to us. This provides us the ability to get feedback from the public, as well as take action on that feedback. For internal development, we use JIRA for story and bug tracking.

2. The data being used in the application is available in JSON format via Swagger API endpoints. These are public and available to anyone.
3. Because the data being used by the application is obtained from the public domain (USGS, etc.), the rights for that data remain as such.
4. Not applicable for this prototype.
5. Because the data being used by the application is obtained from the public domain (USGS, etc.), the rights for that data remain as such.
6. Not applicable for this prototype.
7. We have used Swagger to document our APIs and they are available for third parties to interact with directly. Documentation on the API endpoints can be found here: http://api-cal-notify.symsoftsolutions.com/Swagger/index.html
8. Per the RFI, we have published all the source code on GitHub. See: https://github.com/SymSoftSolutions/cal-notify
9. For this project, all development processes and progress has been made public via the same GitHub repository.

Key questions

1. How are you collecting user feedback for bugs and issues?
2. If there is an API, what capabilities does it provide? Who uses it? How is it documented?
3. If the codebase has not been released under an open source license, explain why.
4. What components are made available to the public as open source?
5. What datasets are made available to the public?

Answers to key questions

1. Initially, we conducted interviews to generate the prototype. Once the product was ready for testing, we conducted additional tests to get feedback and incorporate that into the prototype. Once the prototype is released, users can submit issues and bugs using our open GitHub repository.
2. Yes, an API documented using Swagger is available here: http://api-cal-notify.symsoftsolutions.com/Swagger/index.html. All of the applications core functionality is included in the API and is used by the user interfaces we developed for the prototype. The API is also available to external parties.
3. The codebase has been released as open source through GitHub. See: https://github.com/SymSoftSolutions/cal-notify/blob/master/UNLICENSE.md.

4. The entire codebase has been released as open source through GitHub. See: https://github.com/SymSoftSolutions/cal-notify/blob/master/UNLICENSE.md.
5. All data sources are available to the public (the prototype uses datasets that have already been made public by various federal and state departments).