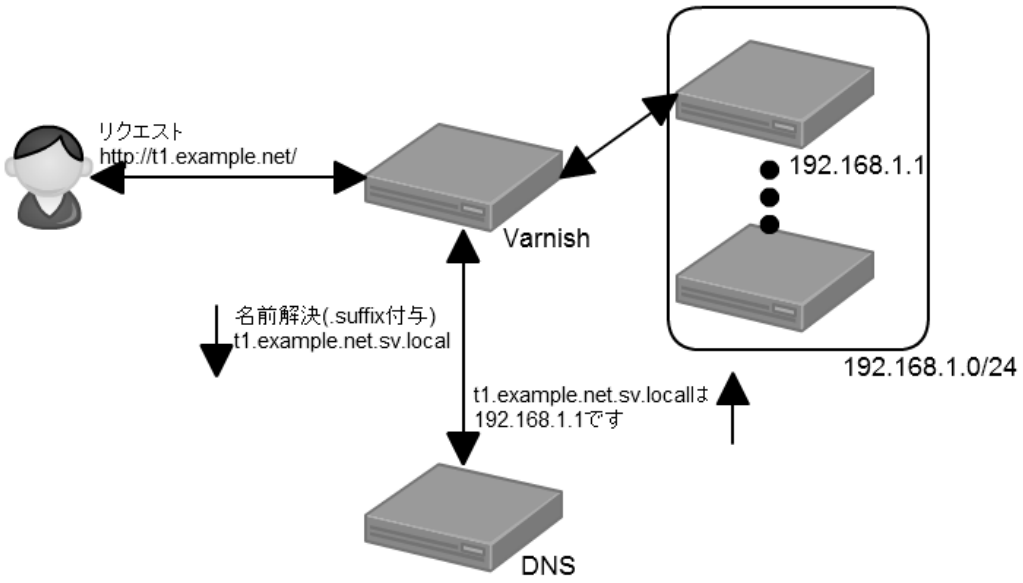


上記図でのディレクターの設定は以下です。

```
director d1 dns{
        .list = {
                .port = “80”;
                “192.168.1.0”/24;
        }
        .ttl = 5s;
        .suffix = “.sv.local”;
}
```

流れは以下です。

1. クライアントが http://t1.example.net/を要求
2. Varnish が t1.example.net を解決するため DNS に.suffix を付与して問い合わせする (t1.example.net.sv.local)
    1. DNS が t1.example.net.sv.local は192.168.1.1と返却
    2. DNS の結果を5秒キャッシュする(.ttl)
3. 192.168.1.1に対して Host を t1.example.net でリクエストする
    1. 結果を受け取る
4. クライアントにレスポンスする

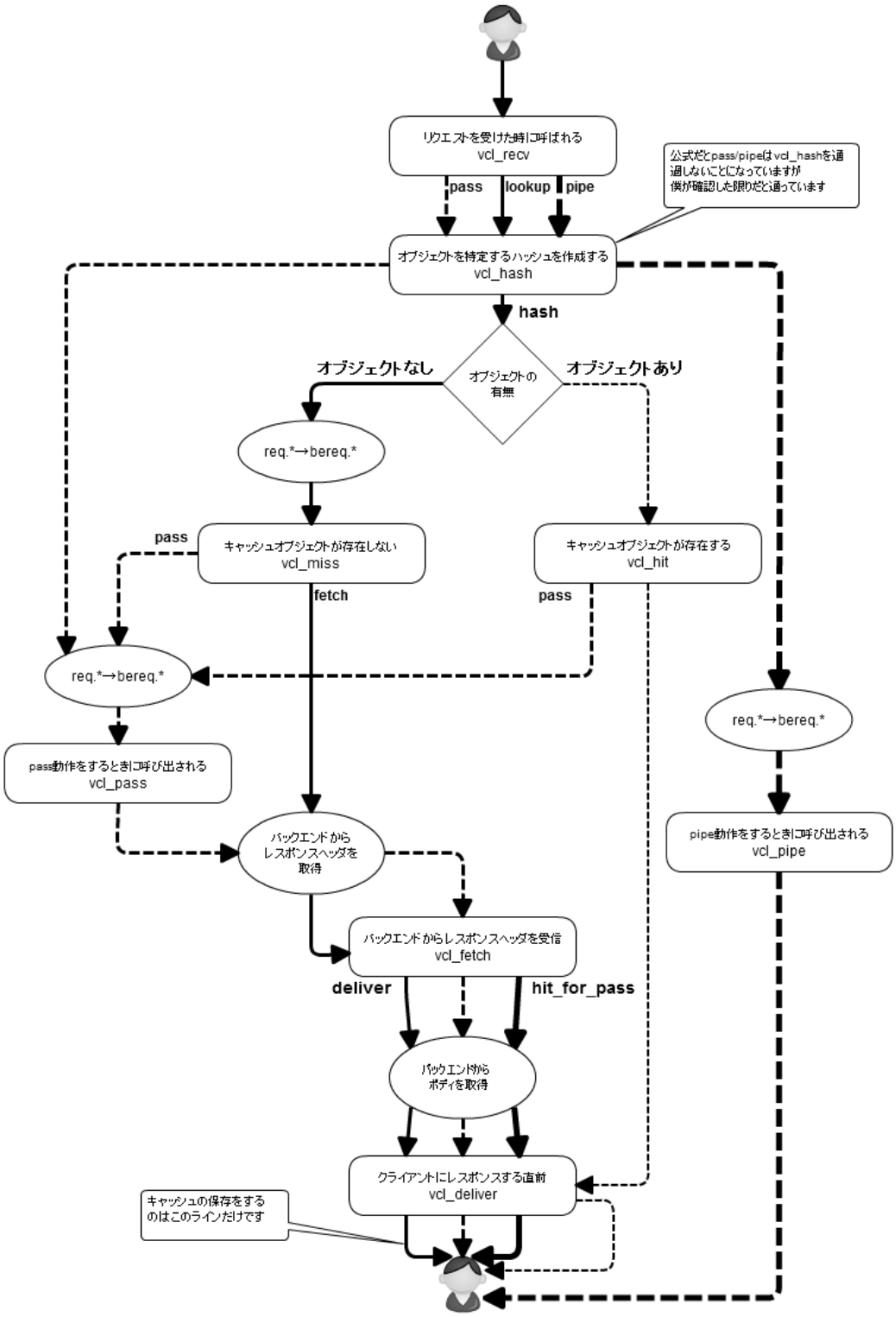
ここで注意が必要なのが名前解決する際は.suffix が付与されますがバックエンドにリクエストするときは付与されません。公式には sv.local の指定だけで動きそうなのですが単純に結合しているので「.」を付けないと t1.example.netsv.local で名前解決しようとするので注意が必要です。

また、DNS が複数の IP を返してきた場合はそれでラウンドロビンします。

なお、.list で範囲指定を行う場合はヘルスチェックができません。そのため backend で範囲分のサーバをインラインで以下のように全て定義してしまうのもいいかと思います。

簡易版アクションフロー図

詳細図は Appendix を参照してください。



り起動できます。

Varnish のデバッグ

何事もデバッグをするにはログを見ることが重要です。しかし Varnish はよくほかのミドルウェアで見るようなファイルには出力しません。一般的にファイルに書き出す処理は非常に重いため高速化のために Varnish では共有メモリに出力しています。
そしてその共有メモリの内容をログ参照用のコマンドを使い見たり、必要に応じてファイルに保存したりします。

まずはデバッグの時によく使う varnishlog について解説します。その他については後述します。

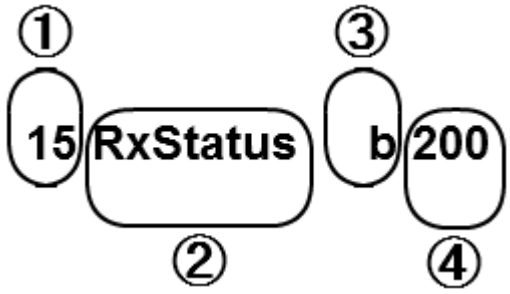
まず Varnish を起動したあと varnishlog を起動します

```
varnishlog
```

特にオプションは必要ありません。そしてブラウザで Varnish にアクセスしてみましょう。そうすると以下のようなログが出力されます。

```
（一部抜粋）
15 TxHeader     b X-Varnish: 1829024782
15 TxHeader     b Accept-Encoding: gzip
15 RxProtocol   b HTTP/1.1
15 RxStatus     b 200
15 RxResponse   b OK
15 RxHeader     b Date: Mon, 18 Jul 2011 15:38:05 GMT
15 RxHeader     b Server: Apache/2.2.15 (Scientific Linux)
15 RxHeader     b Last-Modified: Tue, 12 Jul 2011 13:41:51 GMT
```

パッと見なんとなくわかるようで微妙にわからないと思うので解説します。
基本的にスペース区切りで4ブロックに分かれています。



1. **トランザクショングループ**　**15** RxStatus　b 200
   HTTP トランザクションのグループです。
   同じ番号は同じ HTTP のトランザクションに属します。
   あくまで HTTP なのでクライアント・Varnish 間と Varnish・バックエンド間は別の番号が振られます。
2. **メッセージタグ**　15 **RxStatus**　b 200
   アクティビティの種別でタグが付きます。
   Prefix に Rx,Tx が付いている場合は意味があります。

インラインＣ利用時にヘッダを操作する関数

値の設定

void VRT_SetHdr(const struct sess *sp , enum gethdr_e where, const char *hdr, const char *p, ...)

項番	引数名	説明
1	*sp	spを指定
2	where	操作する対象を指定（HDR_REQ, HDR_RESP, HDR_OBJ, HDR_BEREQ, HDR_BERESP）
3	*hdr	操作するフィールドを指定
4～	*p	挿入する文字列を指定、複数指定の場合は結合する。最後にvrt_magic_string_endの指定必須

戻り値 なし

値の取得

char * VRT_GetHdr(const struct sess *sp, enum gethdr_e where, const char *n)

項番	引数名	説明
1	*sp	spを指定
2	where	操作する対象を指定（HDR_REQ, HDR_RESP, HDR_OBJ, HDR_BEREQ, HDR_BERESP）
3	*n	操作するフィールドを指定

戻り値 指定したフィールドの値

*hdr 及び*n の指定方法は以下の通り

null+フィールド名（コロンも含める）の文字列長を8進法で0パディング2桁

VRT_SetHdr(sp, HDR_RESP, "¥010Expires:", “XXXXXX”, vrt_magic_string_end);

VRT_GetHdr(sp, HDR_RESP, "¥010Expires:");

組み込み関数一覧

関数	説明	利用可能なアクション
ban(式)	指定条件でbanする	
ban_url(正規表現)	指定正規表現と一致するreq.urlの場合banする	
call サブルーチン名	別のサブルーチンを呼び出す	
hash_data(式)	オブジェクトを格納する際のハッシュに追加する	hashのみ
panic(式)	指定メッセージを出力して子プロセスを殺す	
purge	即時にハッシュを削除する	miss,hitのみ
return(アクション名)	returnする	
rollback	変数を初期化してvcl_recvに移動	
set 変数 代入演算子 式	変数に値を設定する	
synthetic 式	レスポンスボディを合成する	errorのみ
remove 変数	unsetと同じ（過去の互換性維持のため存在）	
unset 変数	変数を削除する	
error ステータスコード レスポンス	obj.status,obj.responseを指定してvcl_errorに行く	
regsub(置換対象 正規表現 置換式)	正規表現で最初にヒットしたものの置換を行う	
regsuball(置換対象 正規表現 置換式)	正規表現で最初にヒットしたもの全ての置換を行う	